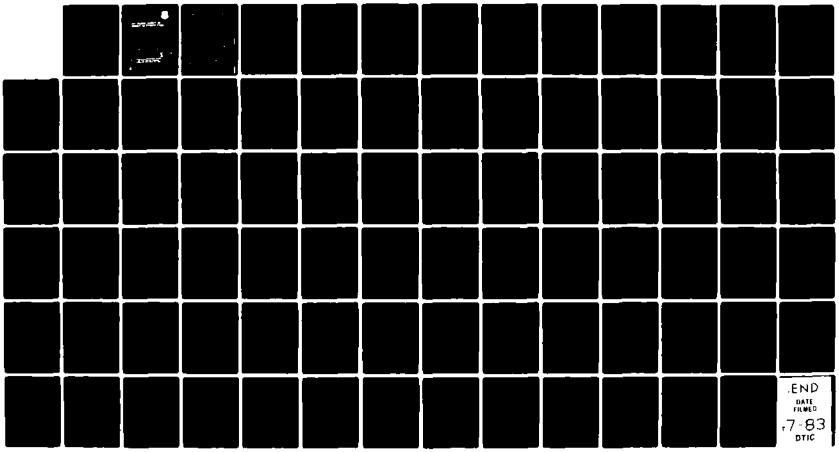
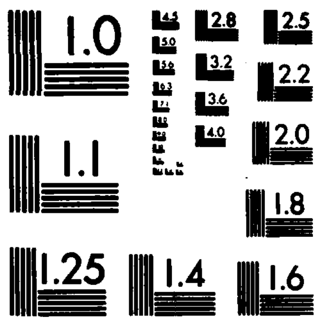



1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

AD A129438

83 06 14 027

[illegible]

[illegible]

[illegible]

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-83-36	2. GOVT ACCESSION NO. AD-A12-9438	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FAULT TOLERANCE, RELIABILITY AND TEST-ABILITY FOR DISTRIBUTED SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Interim Report Sep 81 - Oct 82
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Herbert Hecht Myron Hecht With contributions by K. H. Kim		8. CONTRACT OR GRANT NUMBER(s) F30602-81-C-0133
9. PERFORMING ORGANIZATION NAME AND ADDRESS SoHaR Incorporated 1040 S. LaJolla Ave Los Angeles CA 90035		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 23380245
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (RBET) Griffiss AFB NY 13441		12. REPORT DATE February 1983
		13. NUMBER OF PAGES 84
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Heather Dussault (RBET)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Fault Tolerance Distributed Systems Reliability Improvement Testability	Fault Isolation Fault Tolerant Design Networks Effectiveness	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A growing need exists for improved fault tolerance, reliability, and testability in distributed systems which support Command, Control and Communications and Intelligence (C^3I) activities. The objective of this study is to provide a foundation for the development of design measures and guidelines for the design of fault tolerant systems. Taxonomies of fault tolerance and distributed systems are developed, and typical Air Force C^3I needs in both fault tolerant and distributed computer systems —

are characterized. Reliability and availability experience for ten typical computer systems is reported in a consistent format, and the data are analyzed from the perspective of a distributed system user. Previous work on the identification of problems in distributed systems and design methods for their solutions is discussed. Key issues in the design of fault tolerant distributed systems are identified. Fault location techniques for specific computer configurations found in C^3I applications are described in detail. The study is a continuing effort, and a comprehensive design methodology will be developed based upon the material presented in this report.

SUMMARY

This report covers the first half of a study of fault tolerance, reliability, and testability in distributed systems that support command, control, communications, and intelligence (C3I) activities. The study is motivated by the need for continuous availability of the computing function in the C3I applications and by the increasing utilization of distributed computing in this field. The study is intended to provide a framework for the characterization of fault tolerance provisions, their evaluation against the needs of C3I activities, and recommendations for improvements in fault tolerance, reliability, and testability where these are warranted.

The methodology utilized includes reviews of the general literature of fault tolerant and distributed computing with particular emphasis on reports generated by DoD agencies related to C3I activities; on-site reviews of the reliability experience of selected DoD facilities; collection of pertinent reliability data from non-DoD facilities where these can be obtained; and original research in areas not adequately covered by prior investigations.

As part of the effort reported here, taxonomies of fault tolerance and of distributed systems were developed (Sections 2.1 - 2.3), and functional needs of C3I activities for fault tolerance have been characterized (Section 2.4). The reliability and availability experience of ten typical computer systems (including two Air Force applications) is reported in a consistent format, and the data are interpreted from the point of view of a user of distributed systems (Section 3). A framework for the investigation of design methodologies for distributed systems is described (Section 4.1), and previous work is summarized and key issues in design are identified (Sections 4.2 and 4.3). Fault location techniques applicable to specific computer configurations found in C3I applications are described in detail (Section 5).

The study is continuing, and a comprehensive design methodology will be developed based on the work reported here.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

ACK	(acknowledge) A transmission control character transmitted by a receiver as an affirmative response to the sender
ARPANET	A computer communications network sponsored by the Advanced Research Projects Agency (ARPA)
ATC	Air Traffic Control
AWACS	Airborne Warning and Control System
BMD	Ballistic Missile Defense Command, also ballistic missile defense (in the general sense)
CCITT	Consultative Committee on International Telegraphy and Telephony
CPU	Central processing unit (of a computer)
CRT	Cathode ray tube
CSS	Communications System Segment (of the 427M System)
C3I	(pronounced "c-cube-eye") Command, control, communications and intelligence
EMI	Electro-magnetic interference
ESS	Electronic Switching System (for telephone networks)
FAA	Federal Aviation Agency
FILAN	Flexible Intraconnect Local Area Network (an Air Force project)
FRB	Federal Reserve Bank
I/O	Input/output
ISO	International Standards Organization
NAK	(negative acknowledge) A transmission control character transmitted by a receiver as a negative response to the sender
NCS	NORAD Computer System
NORAD	North American Air Defense Command
PABX	Private automated branch exchange (for telephones)
RADC	Rome Air Development Center

SCC	Space Computational Center
SLAC	Stanford Linear Accelerator Computer
SNA	System Network Architecture (an IBM Corp. concept)
UUT	Unit under test
WWMCCS	Worldwide Military Command and Control System

PREFACE

A growing need exists for improved fault tolerance, reliability, and testability in distributed systems which support command, control, communications, and intelligence (C3I) activities. This interim report identifies those system functional needs, design motivations, and key design issues, and presents a logic which can be used for comparative analysis and evaluation of fault tolerant distributed system reliability, testability, and effectiveness. The results presented are based upon current operational experience and previous studies in the areas of fault tolerant design and distributed computing. This report is intended to provide a foundation for the development of measures and guidelines for the design and evaluation of fault tolerance, reliability, and testability in distributed systems.

SECTION 1 — INTRODUCTION

This is an interim report generated on RADC Contract F30602-81-C-0133, Reliability/Testability/Design Considerations for Fault Tolerant Systems by SoHaR Incorporated. The study is particularly aimed at applications in the command, control, communications, and intelligence area (C3I). At the time of the writing of this report, approximately one-half of the twenty-eight month duration of the project had elapsed. The major goal of the present report is to describe the fault tolerance, reliability and testability found in present C3I systems, or in systems that are technically similar to those in the C3I field.

Distributed systems are coming into increasing use throughout the digital processing field because of the flexibility, performance, and reliability advantages which they offer. Examples of benefits in flexibility are (a) the ability to route computing tasks to the most suitable processor (as contrasted with the local processor that may not be very efficient for a given task), (b) the ability to add processors incrementally as the computing load increases, and (c) the ability to introduce technical advances gradually, one processor at a time, while retaining existing computers on-line, thus avoiding the major software and systems problems that arise when dedicated computers are replaced by newer models. Performance (throughput of computing tasks) is enhanced because distributed computing allows any temporarily idle computer to be utilized for sharing the load at a busy site, and, similarly, reliability is improved because other processors can be utilized to take up the load of a failed one until it is repaired.

All of these benefits are particularly welcome in C3I applications. Major new techniques are being introduced in several functional areas, and the flexibility offered by distributed systems is highly desirable to support a smooth transition to these. The performance advantages are valuable because of the high ratio of peak load to average load, and the resulting oversizing (in terms of average load) that is necessary if dedicated computers are used at each site. The reliability advantages of distributed systems translate directly into survivability, perhaps the most highly prized attribute in a C3I system.

A number of other RADC projects address architectural aspects of distributed systems and configurations for specific applications. The present study is particularly concerned with those aspects of reliability, fault tolerance, and testability that are applicable to broad classes of systems. The definitions and classifications of systems presented here, the experience on current systems. and the techniques described in this report will be analyzed and integrated (together with additional information) during the remainder of this study. The final report will constitute a guideline for achieving reliability, fault tolerance, and testability in the design of distributed systems for C3I applications.

Section 2 of this report introduces the terminology for distributed systems and fault tolerance, presents classification schemes (taxonomies) for both of these concepts, and, in the final part of the section, discusses the objectives and problems in achieving fault tolerance in distributed systems.

Section 3 deals with the reliability of current systems that employ components or techniques that will be applicable to distributed systems in the future. In none of the instances for which data are presented do present systems meet all of the criteria for a truly distributed system that were described in Section 2. Nonetheless, the examination of the current data is essential because it is the basis from which planning for the future must proceed. Due to the cooperation received from a number of Government and private organizations, the long-term (mostly one year) reliability experience of ten systems is presented in a consistent format, with allocation of failures to hardware, software, and other causes. This collection of data may also be of interest to readers outside the field of distributed systems. The final part of Section 3 presents a preliminary interpretation of these data for future C3I systems.

Section 4 discusses design issues and methods in distributed systems. That part of the report is primarily intended to define the constraints within which guidelines for fault tolerance, reliability and testability must be developed. The motivation of the developer/user, the current state of supporting technologies (primarily in networking), and the approaches of established design methodologies are reviewed. The contributions made in the development of specific distributed systems are summarized in the final part of that section.

Section 5 contains an example of a formal fault location technique for several configurations that were repeatedly encountered in current military systems that employ distributed processing of modest scope (a limited number of computers). Fault location is a significant aspect of the testability of distributed systems. Prof. K. H. Kim, a consultant to SoHaR on this effort, originated the concepts used in Section 5 and generated the program design for fault location that is presented in the Appendix. Sections 4 and 5 constitute examples of individual issues and techniques that must be mastered for the successful application of fault tolerance in distributed systems.

SECTION 2 — FAULT TOLERANCE AND DISTRIBUTED SYSTEMS

This section introduces basic concepts of fault tolerance and distributed systems as a foundation for the remainder of the report. Section 2.1 defines key terms that are used throughout the report, section 2.2 contains a taxonomy of fault tolerance measures applicable to distributed systems, and section 2.3 describes the classification of distributed systems. Finally, section 2.4 describes functional needs in fault tolerance and distributed systems.

2.1. DEFINITION OF KEY TERMS

2.1.1. Error, Failure, and Fault

The terms error, fault, and failure are often used interchangeably in technical literature. However, with the introduction of systems that continue to operate when components cease to perform as specified, distinctions among various levels of failures, causes, and effects become necessary. The definitions of these terms used in this report are shown in figure 2-1.

An error exists when the output of a computer system does not meet user requirements, or when the computer is in a state that does not support user needs. The system itself has failed, i.e., execution of a program on this system has resulted in a failure. To cause this failure, a fault must have been present in either the hardware or the software. Hardware faults are frequently caused by deterioration of initially fault-free devices. Because random processes contribute to the deterioration these hardware faults are said to produce random failures. Software faults, as well as hardware design faults, have been present from the time the system was placed into service. They have not resulted in observed errors because of lack of observation or because the external event or trigger to activate them had not been present.

2.1.2. Hardware Fault Tolerance

Hardware fault tolerance is the ability of the system hardware to continue a specified level of operation in the presence of one or more hardware faults. This ability is most often achieved by the use of replicated components. The definition implies that the system must continue to function as specified for all inputs; thus, a system capable of operating in a degraded mode in which a restricted set of inputs can be processed is not fault tolerant.

2.1.3. Software Fault Tolerance

Software fault tolerance is the ability of a system to provide uninterrupted operation in the presence of program faults through multiple implementations of a given functional process. Although this definition was first proposed by Elmendorf more than a decade ago [ELME72], there remains much inconsistency in the usage of this term in the software engineering literature. For example, other techniques such as fault containment and robustness have also been characterized as fault tolerant despite the fact that they do not provide for alternate and independent execution of a function.

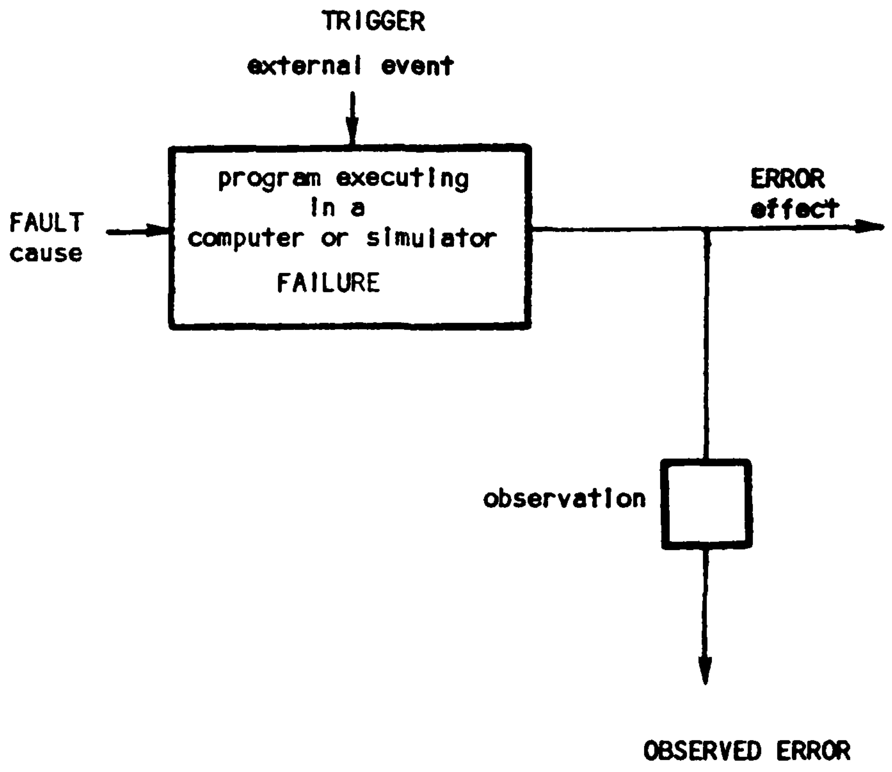


FIGURE 2 - 1 BASIC FAILURE MODEL

2.1.4. Distributed Systems

The term "distributed" when used in conjunction with "processing" or "system" has become one of the vaguest terms in the field of computing. In an introduction typical of many papers on distributed processing, Enslow [ENSL78] spoke of the problem as follows:

> "Words have only one purpose in a technical context — the transmission of information. When they fail to do that, they lead to confusion and misunderstanding. 'Distributed data processing' and 'distributed processing' illustrate that axiom. Like many other words in the lexicon of the computer professional, these have become cliches through overuse, losing much of their original meaning in the process."

Since the publication of that article, an increasing number of vendors, system analysts, and users have adopted the term with a resultant further corruption in its meaning. Thurber's [THUR80] definition of a distributed processing system, which shall be used in this report, consists of a set of six conditions:

1. The system has at least two processors (processing elements; hosts etc)
2. Each processor has a main storage module and other memory subsystems as required.
3. There is no system-wide shared memory
4. There is a communications medium termed the "communications subnetwork"
5. All process communication occurs via messages between processors over the communications subnetwork
6. A message is modeled as a stream of bits broken into three major sections: header, information text, and trailer.

This definition was chosen after examining some of the major Air Force C3I systems in which dispersed computers perform asssociated functions and are linked with various types of communication lines. The dispersed computers may be grouped into tightly coupled networks in which one or more mainframe computers controls an array of sensors, displays, or other devices. Fault tolerance measures can be applied to both the links between these computing centers and within the centers themselves.

Most currently operational large distributed systems consist of a "main" computer installation and "satellite" nodes. The main computer installation contains one or more computers which collectively control the network. Failure of the main computer will result in either a severely degraded or nonfunctional network consisting only of satellite nodes. Each satellite node may have one or more computers which control local (i. e., not connected with other satellites) input and output devices. Failure of one or more of these
network. Thus, failure of any single node results in the loss of some system

processing capability, but does not necessarily result in a total system failure. Each node on a decentralized system may itself be a centralized distributed system. For example, the ARPANET consists of a large number of mainframe computers which control an extensive local network of satellite minicomputers, intelligent terminals, and output devices. None of these nodes, however, controls any other node on the system.

This definition is functional for describing current Air Force distributed systems from the scale of fighter aircraft avionics to the scale of the WWMCCS network. It is also consistent with the current use of the term in the commercial computing industry. Finally, networks which conform to more limited definitions of distributed processing such as [ENSL78 and ENSL81] are also included in this definition.

2.2. TAXONOMY OF FAULT TOLERANCE MEASURES FOR DISTRIBUTED SYSTEMS

Taxonomies for the classification of both fault tolerance methods and network architectures are necessary to partition the topic of fault tolerance in distributed systems into homogeneous and manageable subtopics. The objective of this section is to develop a framework for classifying fault tolerance measures for distributed systems. The basis of this taxonomy is the conceptualization of a computer network as nodes and links. The node is defined as everything on the computer side of the I/O buffer, and the link is defined as the network system beyond the I/O buffer until that of the next node.

Figure 2-2 shows the taxonomy. Fault tolerance for distributed systems can be implemented either with or without reconfiguration of the network. The left hand side of the tree shows fault tolerance implementation with reconfiguration consisting of node substitution, link substitution, or both. Reconfiguration is the highest level of fault tolerance for a distributed system, and requires a network management system. Commercially available protocols and network architectures allow for the reconfiguration (i. e., the disconnection or reconnection while the rest of the network remains unaffected) of secondary network processing elements. However, most work on network reconfiguration after failure of principal processing nodes has been on either a theoretical level or on experimental systems.

The right hand side of the tree shows how fault tolerance is applied without network reconfiguration. In this case, recovery after a failure is achieved by returning all nodes and links to an operational status. Restoration of communication through a link is achieved by one of several strategies: time based (e.g. NAK/ACK) for transient failures, alternate types of communication links for longer term faults (e.g. use of optical communications in the event of extended electromagnetic disturbances), or the use of an alternate route of the link (e.g. a replicated bus running along different sides of an aircraft to mitigate the effects of battle damage). The restoration of a node is achieved through computer hardware or software fault tolerance techniques.

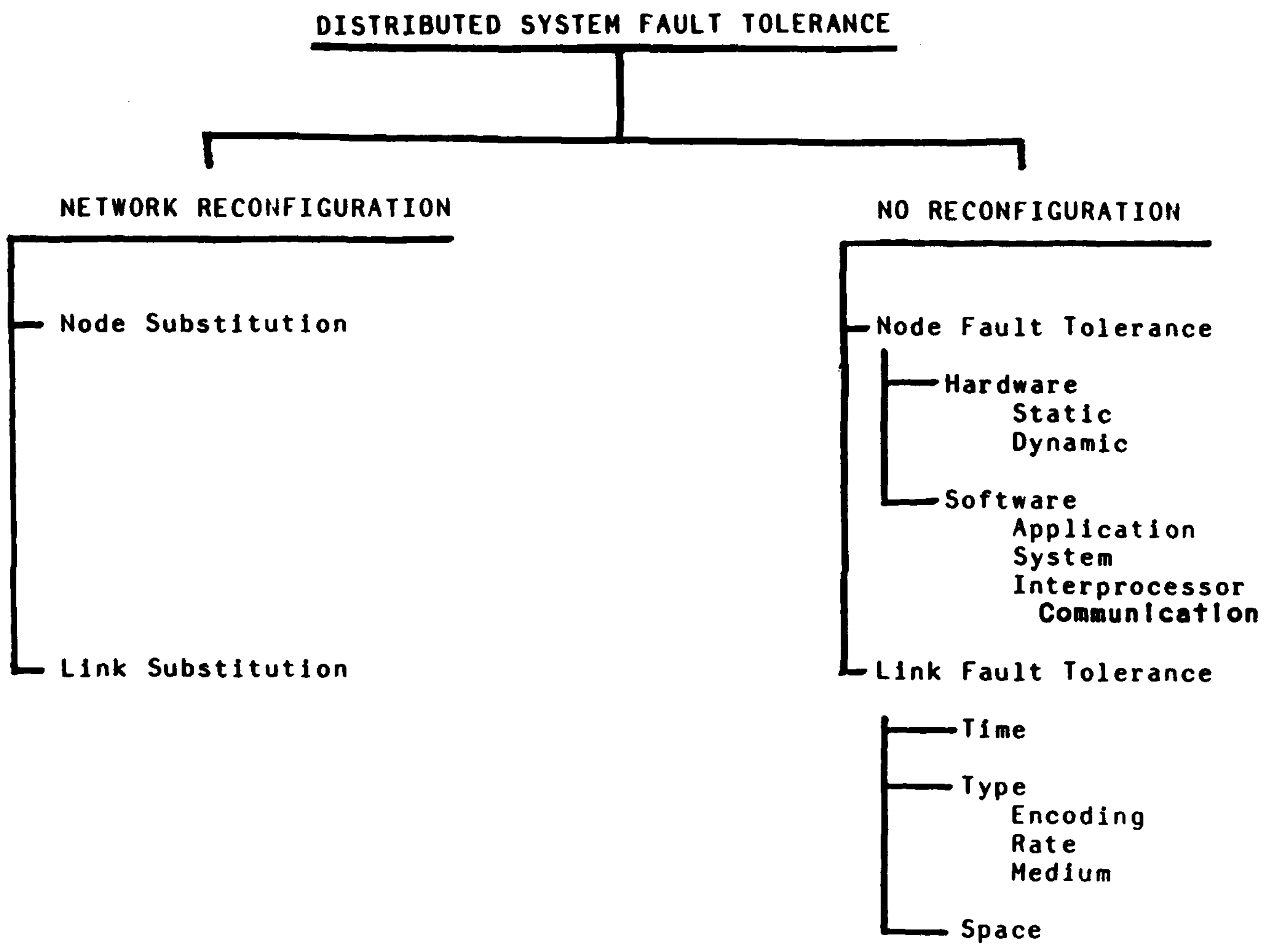


FIGURE 2 - 2 TAXONOMY FOR FAULT TOLERANCE IN DISTRIBUTED SYSTEMS

This division of fault tolerance measures provides a framework for further discussion and analysis, but should not be interpreted as meaning that use of techniques in one classification prevents use of techniques in another class. For example, a fault tolerant computer must have both hardware and software fault tolerance. Similarly, achieving fault tolerant communication links may involve time, type, and space tactics, and may also include use of an alternate link as an additional backup measure.

2.3. TAXONOMIES OF DISTRIBUTED SYSTEMS

The taxonomy used in this work on reliability, maintainability, and fault tolerance characteristics of distributed computer systems was tailored to operational Air Force systems and analogous non-military systems. It utilizes a small number of categories that are well defined and within each of which uniform reliability problems are found and solutions can be applied. Section 2.3.1 describes other taxonomy schemes in the literature and explains why they are unsuitable for the purposes of this work. Section 2.3.2 uses the hierarchical model of network architecture to define two taxonomies. Section 2.3.3 describes the lower level taxonomy (designated as the "network" taxonomy), and section 2.3.4 describes the upper level "application" taxonomy.

2.3.1. Earlier Taxonomies of Distributed Systems

The taxonomy one adopts for distributed systems is determined by the technical point of view. Indeed, so many taxonomies of distributed systems have been presented in the literature that it is possible to develop a classification scheme for the taxonomies [GREE77, BANN81].

Most taxonomies take a topological approach by defining primitives (e.g. nodes, switches, and links) and then classifying the ways in which they can be linked together. The scheme most often cited in the literature using this approach is that of Anderson and Jensen [ANDE75]. The topological view is problematic because other aspects of the network can have more impact on the system characteristics. For example, the computing system of a major Los Angeles newspaper and that of a C3I installation both consist of two replicated mainframe computers and two front end processors. Although these systems are topologically similar, they are very different in most other aspects.

Authors such as Thurber [THUR78] and Jensen, et. al [JENS76] use switching methods (i. e., no switching, circuit switching, message switching, and packet switching) in their classification schemes. This approach fails to consider differences in local and long haul computer networks, and also fails to consider topological and operational aspects of a network. For example, the Ethernet [SHOC82] is topologically a linear bus system (i. e., a variety of nodes are connected to a single communications channel) in which there are no discrete switching elements. Thus, one might place this network in the no switching classification. However, the Xerox implementation of a network using Ethernet is based on fixed length message packets, and it is therefore often characterized as a packet switching network.

Other authors have attempted to address the many aspects of distributed systems by developing elaborate taxonomies. One such classification scheme has five levels and more than sixty categories [BANN81]. Unfortunately, the complexity of this approach makes it impractical.

2.3.2. Taxonomy Used for This Study

The taxonomy for this study regards distributed system architectures as a series of layers, a concept which has been prominent since the development of ARPANET in the late 1960s [KLEI78], [ISO81]. The top layers include the application program and associated display terminal (or other I/O device) control characters. Intermediate levels interface the applications program to the host computer and the intercomputer communication system (often designated the "subnetwork"). The bottom layers control node access to the communication subnetwork and actual data transmission.

Figure 2-3 shows the seven level ISO Reference Model [ISO81] which is the basis of much of the current work in distributed systems. The dotted lines between the levels at the two nodes demonstrate the notion of transparency, i. e., viewing each level as communicating directly with its counterpart at the receiving node without regarding the intervening levels through which the data actually passes during the transfer. The desired result is to decouple application programs and data from lower levels which control the actual operation of the network. Figure 2-4 shows how these layers are aggregated for this study. Reliability, maintainability, and fault tolerance characteristics of either grouping of layers will be considered separately, with the resultant development of two corresponding taxonomies.

Because the lower layers of figure 2-4 are affected by the nature of the network, (e.g. local or long haul, transmission rate and medium, performance characteristics of the node hardware, etc), the taxonomy for this level is designated as the "network" taxonomy and is discussed in subsection 2.3.3. The upper layers are related to the particular applications tasks and data, and the associated taxonomy is therefore designated the "application level" taxonomy, which is described in sections 2.3.4.

2.3.3. Network Taxonomy

Figure 2-5 shows the overall structure of the network taxonomy. The left branch includes three types of networks confined to a small physical area and the right branch describes two types of dispersed networks. The distinction between these branches is the ratio of the communication link bandwidth to the computing node processing throughput, a quantity which governs the efficiency with which computing nodes can interact. In localized networks, where links have capacities in the Megabit per second and higher range, this ratio is generally on the order of a few percent. In dispersed networks, where long distance links normally have capacities of less than 20,000 bits per second, it is several orders of magnitude lower.

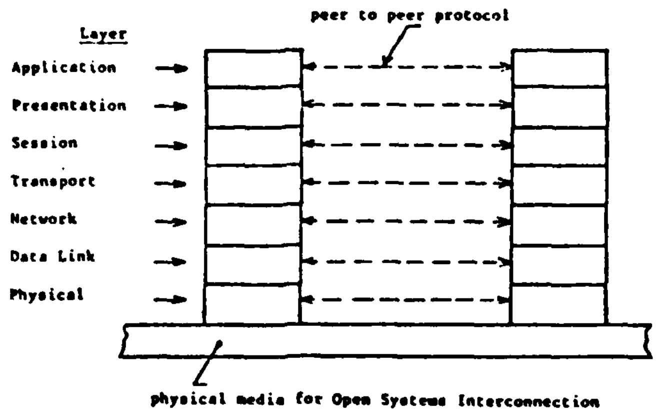


FIGURE 2 - 3 ISO TC-97 OPEN SYSTEMS INTERCONNECTIONS LAYERS

NETWORK LAYERS	
7 APPLICATION	APPLICATION LEVEL TAXONOMY
6 PRESENTATION	
5 DATA FLOW	
4 TRANSMISSION	NETWORK LEVEL TAXONOMY
3 PATH	
2 LINK	
1 PHYSICAL	

FIGURE 2 - 4 TAXONOMIES AND CORRESPONDING ISO TC-97 NETWORK LAYERS

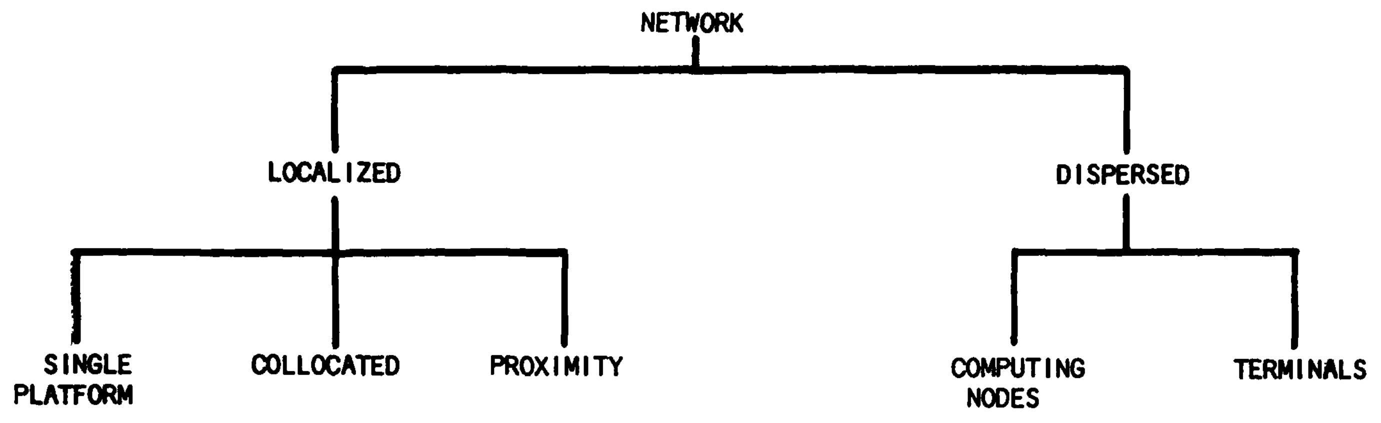


FIGURE 2 - 5 NETWORK LEVEL TAXONOMY

The three subclassifications of the localized computer networks shown in figure 2-5 are mobile, collocated, and proximity computer networks. The latter designation refers to a network whose nodes are located within a radius of approximately 5 km (the predominant definition of local computer networks for office automation purposes).

Because of size and weight limitations, mobile localized networks generally consists of mini and microcomputers. The distinguishing features of this category are the limited maintenance and diagnostic resources which may be applied during operation, timing and program length constraints, and the time critical nature of interruptions and recovery procedures. The major operational example of an Air Force system in this classification is AWACS. A developmental microprocessor based system currently exists at Wright Patterson Air Force Base [LAR181]. Network reliability problems include failure detection, isolation, and reconfiguration due to either component malfunctions or battle damage. These functions must be performed both automatically because human operators may be fully occupied with other tasks and rapidly because of the real time applications.

Networks of collocated computers, the second subclassification, are fixed ground based computers interconnected to achieve higher system reliability, throughput, or task integration. Systems in this category are distinguished from the previous one by fixed locations and the resultant relaxing of constraints on component weight and size, diagnostic provisions, and maintenance capabilities.

Reliability problems for this category of distributed systems include local failure detection, isolation, and reconfiguration. In most cases, links between the computers do not contribute significantly to the network failure rate.

Distributed systems in the proximity subclassification are ground based networks in which nodes are located in the same general vicinity but are not physically adjacent. These systems are currently designated as "local area networks". The major distinctions of this category are the internode distance and the use of serial (rather than parallel) communication on the links. Two examples of Air Force systems are the FILAN specification now being developed at RADC [FILA82] and Ballistic Missile Defense (BMD) systems now being developed by the U. S. Army [ALF081]. A third example is the combination of computers on an AWACS aircraft, on fighter aircraft being controlled by the AWACS, and at a ground station. Network reliability problems for these systems include remote failure detection and isolation (the malfunctioning node may be inaccessible because of distance or battle considerations), reconfiguration, and disconnection of a "babbling node". As a consequence of the increased internode distance, problems on the link related to noise and signal propagation time must also be considered.

Figure 2-5 shows two subclassifications for dispersed computer networks: (1) dispersed computers, i. e., a network with large computers scattered over a wide geographical area, and (2) dispersed terminals, i. e., a network with one or more computers located at a central site which support sensors, terminals, and other specialized devices over a wide geographical area.

The prime military example of the first subclassification is the Worldwide Military Command and Control System (WWMCCS), a system which includes sites in Europe, North America, the Pacific, and Asia [GAO78]. Other example systems which were surveyed are shown in table 3-1. Network reliability problems include ensuring the integrity of communication links to other computers, error detection and correction of the transmitted data, remote failure detection and isolation of both computers and communication links, and establishment of alternate links to disconnected nodes.

The distinguishing characteristic of the second subclasssification of dispersed systems is the presence of geographically separated terminals (and other I/O devices) and a central computing facility. If the computing facility contains more than one local computer, that part of the network falls into the localized classification while the portion concerned with the remote terminals falls in this category. Military examples of such systems include NORAD and PAVE PAWS [GAO78] which have both multiple collocated mainframe computers and links to remote sites. Network reliability problems include ensuring the integrity of the communications link to the terminals, error detection and correction of transmitted data, remote failure detection and isolation of communication links and terminals, and rerouting of communications to critical disconnected terminals.

2.3.4. Application Level Taxonomy

Figure 2-6 shows the taxonomy for the application level. The two major divisions are based on the need for shared programs and data among two or more computing nodes. The left branch of the taxonomy comprises those applications which do not involve shared programs or data. The right branch consists of two classes of shared programs or data: replication and partitioning. Partial replication is a special case of partitioning.

The primary military example of a computer network falling into the unshared subclassification is a single AWACS aircraft. The navigation, communication, display control, and central computers perform unrelated tasks and, although the first three computers interface with the fourth, there are no common programs or data. A second military example is the NORAD computer complex [GAO78] in which three separate computer systems perform distinct but interrelated functions. Network reliability problems in this category are task scheduling after reconfiguration (if it is possible to reallocate tasks from a failed node onto working nodes), network recovery on the application level, and interprocess communication for both co-resident tasks and those resident on different processors.

PAVE PAWS is the best example of a computer network in which shared programs and data are replicated. Network reliability problems at the applications level include updating procedures and concurrency control, network recovery, and interprocess communication. Because of the complexity of implementing partitioned distributed programs and data bases, no examples were included in this study. Network reliability problems are concerned with interprocess communications, concurrency control (i. e., assuring that a READ request is honored only after all earlier WRITES are performed), programs and data redundancy measures, and network recovery.

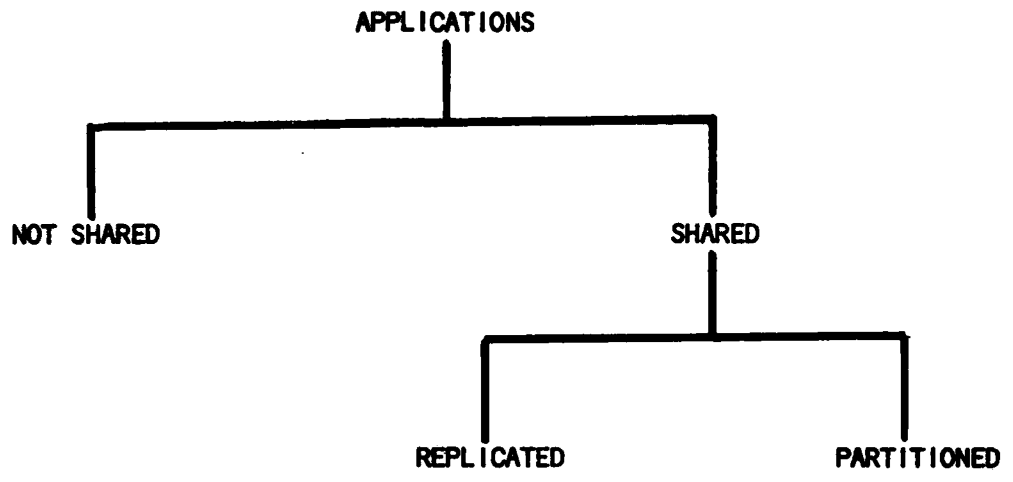


FIGURE 2 - 6 APPLICATION LEVEL TAXONOMY

2.4 FUNCTIONAL NEEDS FOR FAULT TOLERANCE IN DISTRIBUTED SYSTEMS

This section describes typical Air Force C3I needs in both distributed computer systems and for fault tolerance in these systems. Because the aim of this study is to advance the state of the art, this section emphasizes needs that are not currently being met. However, it should be noted that some fault tolerant capabilities in distributed computing do exist at present.

2.4.1 Functional Needs in Distributed Computing

There is a pervasive need in C3I applications to access programs and data from remote files or real-time data sources, to combine these with local programs and data, and to cause actions to be taken on output derived from the combined data. A typical case is the identification of the launch point of an incoming missile from (a) a file of potential launch sites that may be in a local data base and (b) track information that is coming in from one or more remote sites, and then to notify affected commands of the results of this identification. When data sources have been selected in advance, this computation can proceed without operator involvement, and the results placed on hardwired communication links. However, when the situation demands a more general solution, it is desirable that an operator on a properly privileged terminal be able to set up an equivalent computation by means of the computer statements shown in figure 2-7 (A).

Ideally, the operator need only identify the desired procedure (DATAMERGE), the type of data (A and B), and the disposition of the output (creation of a file MERGED). The distributed computer system will then (1) select the most suitable and available computer for this procedure, (2) access the most current sources for data A and B, and (3) store the resultant file in the most accessible device. This capability is not implemented in currently operational systems.

Instead. the operator is forced to select a computer, to identify sources for the programs and data, and to tell the system where to store the result as indicated in figure 2-7 (B). In routine situations these operator actions are trivial, and a strong argument can be made that the ability of current systems to automate the access (item 2 in the previous paragraph) is a major achievement. However, what if the routinely programmed computer for this procedure is already fully loaded, the routinely accessed programs and data sources have not been updated (but another source has been), and the routine storage device is down or does not have sufficient capacity for this file? All of these difficulties are much more likely to arise in exactly those situations when C3I systems must perform 'for real'.

Therefore, a substantial incentive exists for achieving the capabilities of figure 2-7 (A). A major problem is the tendency of present support software, particularly the compilers, to bind an application to a specific computer. *Typical application programs can only be run on one specific type of computer* after compilation into object code as indicated in figure 2-8 (A). Even a routine modification such as adding memory will require recompilation in many cases. In order for a distributed system to assign application programs to any available computer, it is necessary to separate those portions of the compilation which translate source code from those that provide the computer adaptation as shown in figure 2-8 (B). While there are tendencies in that direction, much more effort seems necessary to meet the functional needs of C3I users.

2.4.2 Functional Needs for Fault Tolerance

An analogous situation to that of distributed programs and data exists for fault tolerant features in distributed systems. Ideally, the operator should be able to indicate simply that fault tolerance (or perhaps a specific degree of fault tolerance) is desired, and the computer system should then configure itself to provide the required back-up elements as implied by the instructions shown in figure 2-9 (A). However. the best available technology toward this end is in fixed redundant installations with the ability to automate back-up programs and data storage (not always efficiently). In case of a computer failure, the user must select the alternate, purge files that may contain improper programs and data, and identify a suitable restart point as indicated in figure 2-9 (B).

There are few specific obstacles to achieving the desired fault tolerance capabilities once the problem of assigning suitable alternate computers has

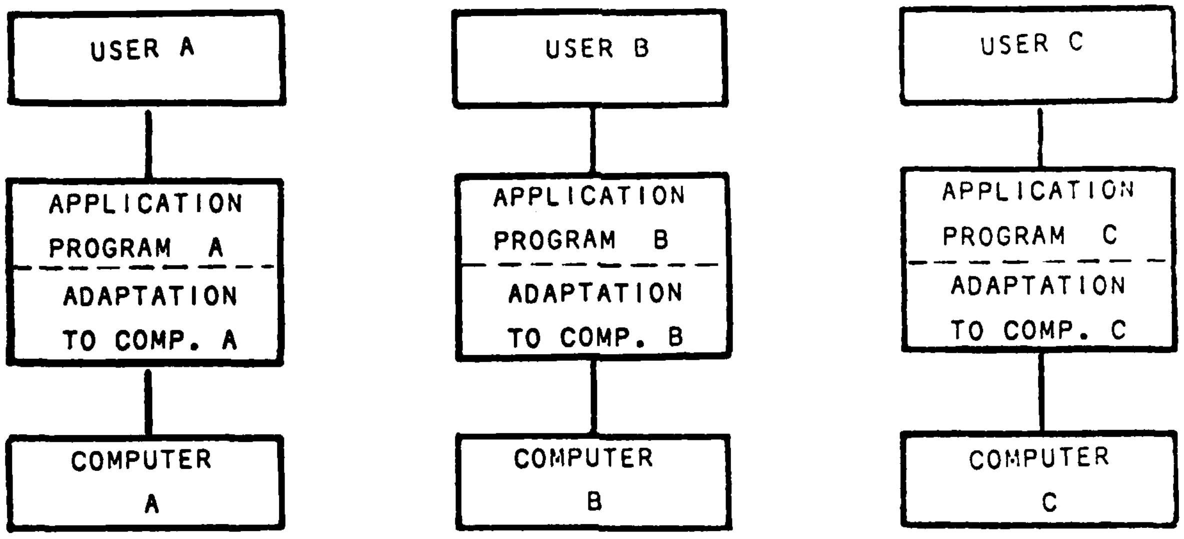
```
PROCEDURE  DATAMERGE
        DATA A, DATA B
        SAVE MERGED
END PROCEDURE
```

A. IDEAL

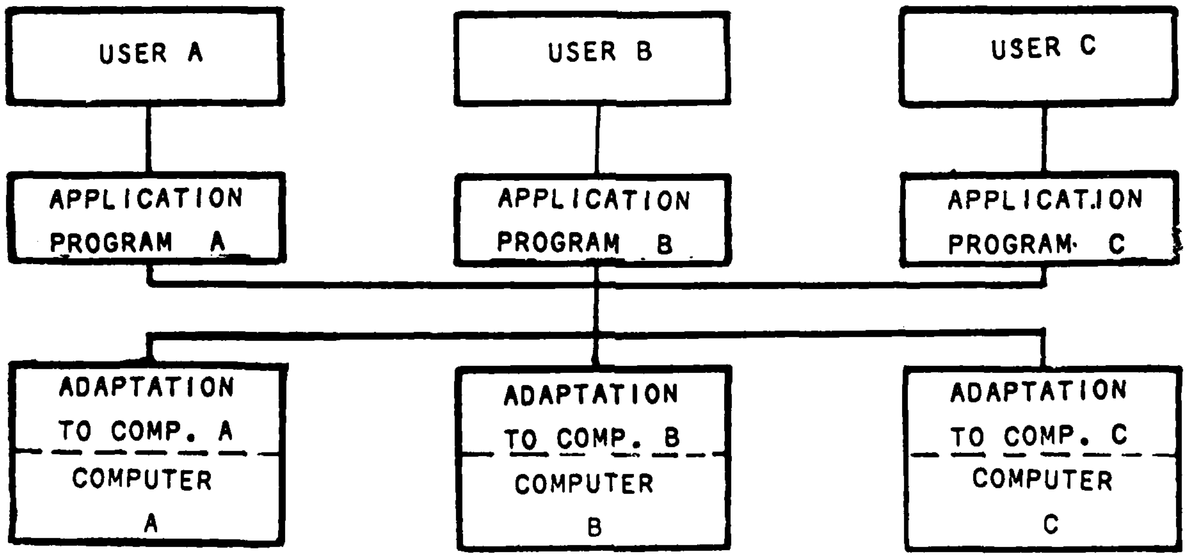
```
CONNECT COMPUTER ALPHA
PROCEDURE  DATAMERGE
        DATA A ON X, DATA B ON Y
        SAVE MERGED ON Z
END PROCEDURE
DISCONNECT COMPUTER ALPHA
```

B. BEST CURRENT PRACTICE

FIGURE 2 - 7 FUNCTIONAL NEEDS IN DISTRIBUTED DATA PROCESSING



A. DEDICATED COMPUTER APPROACH



B. DISTRIBUTED COMPUTING APPROACH

FIGURE 2 - 8 BINDING OF APPLICATIONS TO COMPUTERS

```
PROCEDURE DATAMERGE (FAULT TOLERANT)
       ...               ...
       ...               ...
END PROCEDURE
```

A. IDEAL

```
CONNECT TO ALPHA
PROCEDURE DATAMERGE
 ...          ...
  SAVE MERGED ON Z AND ZZ
END PROCEDURE
```

IN CASE OF FAILURE, CONTINUE:

```
CONNECT TO BETA
PURGE DATA ON Z AND ZZ
PROCEDURE DATAMERGE
 ...          ...
  SAVE MERGED ON Z AND ZZ
END PROCEDURE
```

B. BEST CURRENT PRACTICE

FIGURE 2 - 9 FAULT TOLERANT DISTRIBUTED DATA PROCESSING

been solved. Thus, an improvement in the functional capabilities relative to distributed computing will pave the way for a significant improvement in practical fault tolerance.

SECTION 3 — RELIABILITY OF CURRENT SYSTEMS

The reliability experience on current systems represents a starting point for what might be expected for future systems and for determining the types of reliability improvements that would be most effective for these. The first part of this section presents data on ten current systems. the second part analyzes the data, and the third part evaluates the outlook for future systems based on the current experience.

3.1 CURRENT EXPERIENCE

As part of this study, reliability and availability data on ten current systems were obtained in a consistent format. All of these systems serve applications in which it is important that computer services be continuously available throughout a specified portion of the day, in some cases for 24 hours, and therefore all of them incorporate redundancy for at least a portion of the local computer installation. None of them uses resources at another node to substitute for failed or overloaded local resources, and in this regard they are not representative of the operation of future distributed systems. Expectations about the reliability of distributed systems are derived as extrapolations from the experience discussed here and are presented in the last part of this section.

The systems for which reliability data were obtainable span a wide range of applications, from telephone switching systems to airline reservations and banking. The systems are not comparable in terms of complexity. In particular, the diversity of tasks handled by the FAA en route air traffic control system makes this a uniquely complex application area. In some cases availability or reliability goals had been established whereas in others it was intended to provide the best service possible. Any grouping is somewhat arbitrary, and comparisons between systems must take into account the wide differences in requirements, development and procurement constraints, and operational practices. The data are presented to show that:

a. availability data are being collected in a consistent format in a variety of applications

b. several systems are available more than 99% of their expected operating time

c. the causes of failures are fairly similar, and are distributed about evenly between hardware, software, and other classifications.

Four of the systems exist in almost identical form in many locations, whereas the others are singular installations. Table 3-1 shows the downtime and related data for systems that are installed in multiple locations.

TABLE 3 - 1 EXPERIENCE OF MULTIPLE INSTALLATION SYSTEMS

System	Bell No. 4 ESS	FAA En Route ATC	Federal Reserve Bank Dataphone 50	Federal Reserve Bank Medium Speed
No. Installed	55	20	13	14
Op. hrs/yr.	8760	7665	3000	3000
Availab. goal	99.99%	-	96%	98.5%
Actual availab.	99.99%	99.6%	99%	98.8%
Downtime				
Avg. hrs/yr	0.75	30	30.3	34.9
Caused by				
Hardware	25%	40%	38%	39%
Software	35%	30%	35%	51%
Other	40%	30%	27%	10%

The Bell No. 4 Electronic Switching System is intended to operate 24 hours every day of the year; the En Route Air Traffic Control System is shut down for maintenance approximately 3 hours each day during the early morning hours and a back-up system is then used to handle the light traffic load; both Federal Reserve Funds Transfer Systems operate approximately 12 hours a day on weekdays only.

Several availability requirements have been established for the No. 4 ESS. One of these is that the average downtime for an installation shall not exceed 6 hours over a 40 year operating life (corresponding to an availability of 99.9983%). Other requirements are specific to the application, dealing with the number of calls that may be interrupted and with the number of unit replacement actions [DAVI81]. The availability requirement for the FRB Funds Transfer System relates to the availability of each installation for a given month. The actual availabilities are in each case averages over all installations for a calendar year. The Bell and FAA actual availability data are for 1980, the FRB actuals are for 1981.

Criteria for downtime are that the entire installation becomes inoperative or more than a threshold amount of time (in the case of the FAA this is one minute; it is less for the other systems in Table 3-1). Partial outages that affect only a limited number of phones, or a single controller's console, are not included in these statistics. Note particularly that failure of a single computer will typically not result in downtime because back-up is available.

The availability experience of six systems that exist in only a single installation is presented in Table 3-2. Two organizations contributed data on two systems each. In one case the two systems used exactly the same hardware configuration but differed in operational details; in the other case there was only gross similarity of equipment. The availability goal for each of the airline systems was 99.6%. Goals for the other installations were not stated. The computer applications represented in Table 3-2 differ greatly in the complexity of the programs, size of data bases, number of access points, and requirements for real-time output. Differences in availability or downtime therefore do not indicate that one system is "better" than another. The data for the airline systems pertain to 1980. All other data represent 1981 experience.

TABLE 3 - 2 EXPERIENCE OF SINGLE INSTALLATION SYSTEMS

System	Airline		Military		Stanford	Gov't
	Flt. Inf.	Reserv.	Syst. A	Syst. B	Lin. Accel.	Fiscal Syst.
Op. hrs/yr.	8760	8760	7835	8630	8518	6535
Actual availab.	99.89%	99.65%	99.22%	98.40%	98.66%	88.52%
Downtime						
Hrs/yr	9.5	31	61	138	114	750
Caused by						
Hardware	23%	41%	28%	49%	56%	64%
Software	16%	35%	10%	1%	35%	14%
Other	61%	24%	62%	50%	9%	22%

3.2 ANALYSIS OF CURRENT DATA

Even the most casual review of the data presented in Tables 3-1 and 3-2 identifies the Bell No. 4 Electronic Switching System as having exceptionally high availability and correspondingly low downtime. This system is the product of a specialized organization comprising several thousand professionals, and, as the designation indicates, it is the fourth major design of an electronic switching system undertaken by that group. Publications on the No. 1 ESS go back at least to 1964 [KEIS64], and features of the No. 4 ESS were described as early as 1972 [VAUG72]. The data in Table 3-1 indicate that the long-term allocation of resources to ambitious and well-specified reliability goals produces the desired results.

Like its predecessors, the No. 4 ESS incorporates dual digital processors and error correcting code in memory. Redundancy is incorporated in peripherals such that a single failure can not disable more than a small number of lines. In 1980 the average installation served 22,000 terminations. The computer program comprised over 2 million instructions [DAVI81].

In the early reliability planning for electronic switching systems it was assumed that most system failures will be caused by simultaneous failures of redundant hardware components, such as a second processor failing while the first one was being repaired. Such incidents accounted for only 11% of all failures and 9% of the downtime in the data reported in Table 3-1. The balance of the hardware downtime was due to wiring failures or errors (6%), necessary shutdowns for fault isolation (6%), and design errors (4%). Software failures were the largest single cause of downtime. They accounted directly for 29%, and they required shutdowns for intentional test, etc. that caused another 6% of the downtime. The largest contributor to the "other" category for the ESS was personnel errors which accounted for 24% of downtime. Unresolved or unclassifiable problems accounted for the balance of the downtime reported in that category. No outages due to power supply problems were reported for ESS. This is in sharp contrast with the experience on other systems.

The most significant facts emerging from the analysis of the ESS data are:

a. the unusually high availability of this system

b. the small contribution to downtime from classical failure mechanisms

c. the importance of software and personnel failures

The FAA en route air traffic control system utilizes computers that are derived from the IBM 360 series and incorporate a very effective error detection and reconfiguration mechanism. Depending on the workload at each air traffic control center, three or four mainframes are provided of which at least one is a spare that is activated in case of a hardware failure in one of the other units. The equipment is representative of computer design in the early 1960's and was installed between 1967 and 1972 [GRAY80]. Although the same hardware and basic software are used in each traffic control center, local modifications and adaptations are authorized to permit each center to meet its local needs. This, in addition to the varying traffic loads, may account for differences discussed in the following paragraph. It also needs to be stated that outage of the computer system does not mean cessation of air traffic control operations at the affected center. There are further back-up provisions which impose a higher workload on the controllers but permit safe handling of controlled aircraft.

The data available on the computer failures of the en route air traffic control system permit some analysis of the differences between centers. The following discussion pertains to the number of failures (hardware, software and unknown, but excluding personnel errors) for the main computer installation at each center. Number of failures rather than downtime was selected so as to exclude (as much as possible) differences in maintenance proficiency, and personnel errors were deleted for the same reason. The average number of interruptions due to the selected causes during 1980 was 162.7 with a standard deviation of 69.2. The lowest number of interruptions observed was 15 and the highest number was 357. Fifteen of the twenty centers (75%) were within one standard deviation of the mean (compared to 65% for a theoretical Normal distribution). Busy air traffic control centers were represented among those with a low interruption frequency as well as among those experiencing an above average number of interruptions. Because workload measures were not available, no formal correlation between traffic volume and failure frequency was undertaken.

Much of this difference between centers must be due to controllable causes (maintenance and administrative practices, nature of the local adaptations, etc.), and it is interesting to speculate how much benefit might be derived from an attack on those causes. If the average frequency of interruptions could have been reduced to the low end of the central range (observed mean minus one standard deviation), and if downtime is proportional to the number of interruptions, then the average annual downtime would have been reduced to less than 20 hours. This reduction is greater than that which could reasonably be expected from reliability improvement programs in either hardware or software.

Despite the age of the equipment and the complexity of the computational tasks, the FAA en route air traffic control computers achieved an availability of 99.6%. The above analysis suggests that this figure could be further improved by control of maintenance and administrative practices.

The Federal Reserve Bank operates two computer data systems: The Dataphone 50 system which is concerned with bulk processing and transfer of computer data (economic analyses, member bank status reports), and the medium speed system which handles individual fund transfer activities. The Dataphone 50 system was inaugurated in 1975, and it operates at 50 kilobaud over a dedicated coax cable. It facilitates point-to-point data transfer between all nodes. The medium speed system has been in operation since 1970. It is laid out as a star network with the central node at Culpeper, Virginia. Its nominal transmission rate is 2.4 kilobaud, and it uses a store-and-forward protocol. A variety of computers are installed at each of the Federal Reserve Banks and have access to either network.

Two interesting observations were made possible on the basis of the material furnisned on the Federal Reserve Communications System: Outages of the central node contributed only a minor portion of the total downtime, and workload did not seem to have a significant effect on the duration of the outages. The Culpeper installation, which serves as the central node for the medium speed system had a total downtime of only 13.5 hours during the year, compared to an outage of 34.9 hours for the average node. Of the 103 failures at the central node, 87 were caused by software problems. Ten failures were due to hardware problems, and six of these were reported during one month, apparently a single problem that was difficult to diagnose. The conclusion from these observations is that hardware fault tolerance at a central node can contribute significantly to the reliability of a star network, but that it needs to be supplemented by software fault tolerance techniques in order to obtain the full benefit of this link structure.

Several investigators have recently reported a strong correlation between workload and computer failures [BEAU79, CAST81. IYER82]. In the data on the medium speed system, downtime during the peak hours for that system (1 pm to 4 pm Eastern Standard or Daylight Saving Time) is stated separately. The average outage per location reported during peak time was 7.05 hrs/yr. Since the average outage for the entire 12 hour operating period was 34.9 hrs/yr, this indicates that less than one-quarter of the downtime occurs during that quarter of the operating day during which the workload is highest. While downtime and failure frequency are not the same, one expects approximately the same fraction of each to occur during a given time interval unless special circumstances prevail. Some explanations for the deviation from the generally expected relation between workload and failure frequency are:

a. Maintenance and staffing schedules favor availability during the peak period. Maintenance actions which might reduce equipment availability during the peak time are avoided. The most experienced operating and maintenance personnel are at work during the busy period. Special procedures are in effect to minimize the probability of a failure during the peak hours.

b. The designation of the peak period may be in error. The workload analysis might have been conducted at some time in the past when a different pattern prevailed. Users may deliberately schedule most of their work during 'off-peak' hours, thereby making these de facto peak hours.

c. The reported relations between workload and failure frequency may not apply. Previous studies have been primarily concerned with processing bound applications whereas the medium speed system is probably channel bound. Effects that have not yet been identified may cause a deviation from the expected pattern.

All three factors might be at work, but on the basis of the procedures followed in similar systems the major contributor to the observed effect is probably (a). The data on the Federal Reserve Communications System show that with proper design and procedures the central node in a star network need not be the weak link, and a disproportionate fraction of the downtime need not occur during the busiest period.

Data obtained in 1976 on the Stanford Linear Accelerator Computer (SLAC) show a very pronounced dependence of failure frequency, particularly for failures due to software, on workload, and this relation is also evident in the current data. Figure 3-1 illustrates the software failure frequency (total for 1976) during each one hour period of the day. Note the peak between 11 am and 12 noon, then a decrease during the lunch period, and a secondary peak lasting from 2 to 4 pm. These are obviously the periods of highest activity on the system.

Only failures which affected the entire system are included in Figure 3-1, primarily failures in the operating programs. Because these programs are particularly active when a new job is started, we normalized the failure frequency relative to the number of job arrivals during each one hour period. The resulting graph is shown in Figure 3-2. The peaks during the mid-day period have been eliminated, and instead there is a pronounced singular peak between 7 am and 8 am. During this period not many new jobs are started, but there is a high degree of system activity due to archiving, re-initialization of the computer, and sometimes phasing in a new release of the operating system.

The Government fiscal system described in the last column of Table 3-2 consists of a redundant installation of IBM 370/168 computers that service a

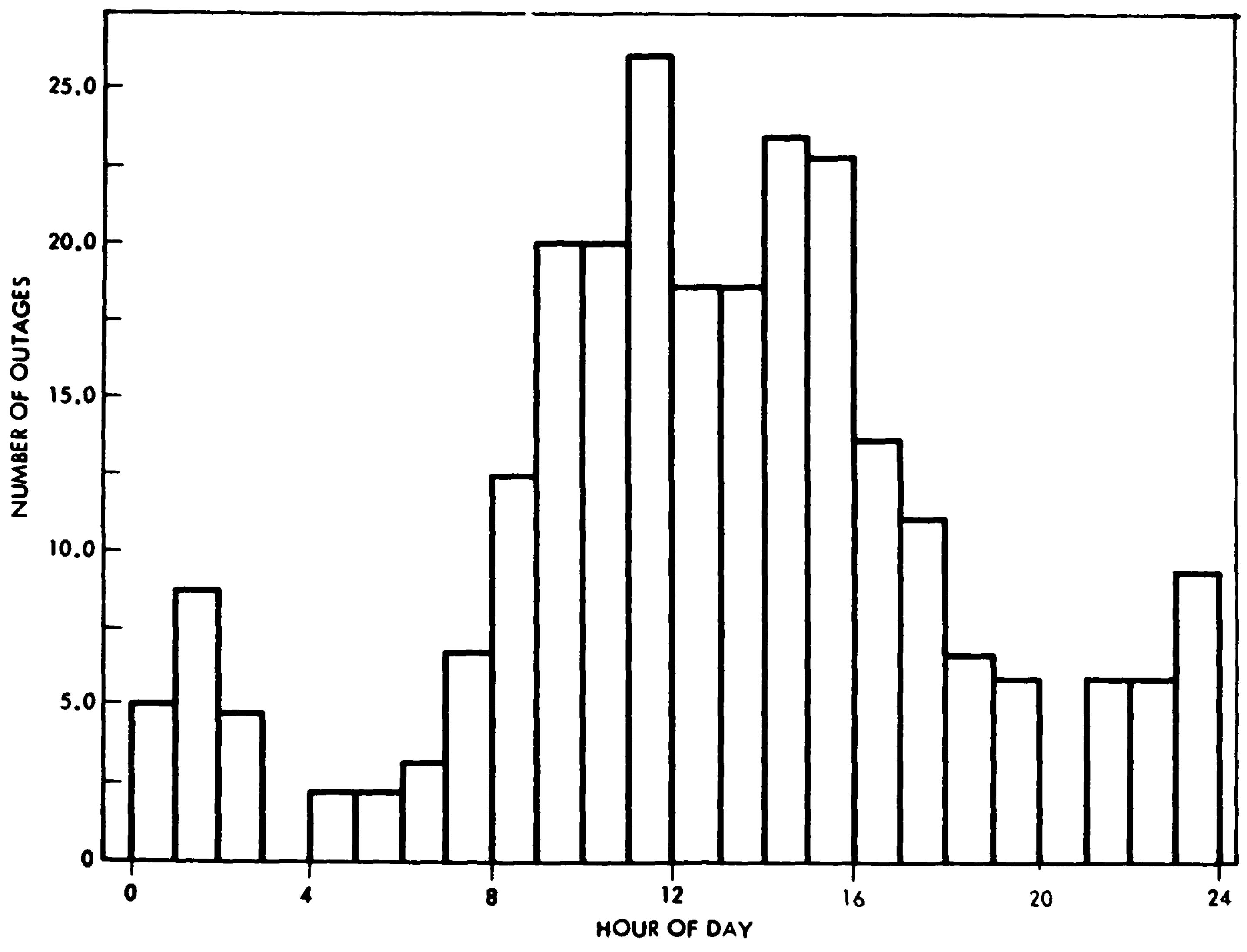


FIGURE 3 - 1 SLAC COMPUTER OUTAGES DUE TO SOFTWARE FAILURES

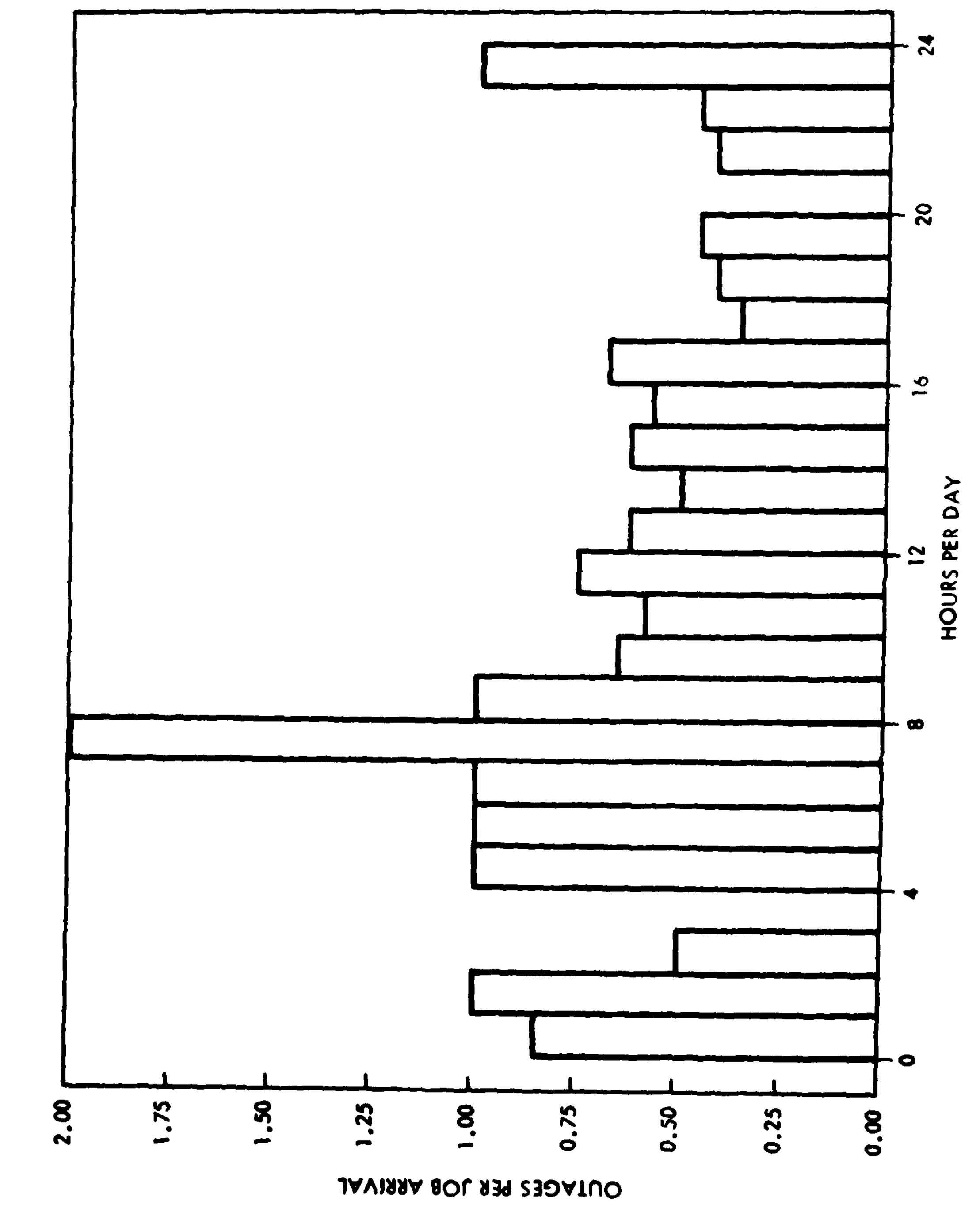


FIGURE 3 - 2 SLAC COMPUTER OUTAGES NORMALIZED TO JOB ARRIVALS

nationwide network of approximately 1,000 terminals that direct queries to the central data base and can also, with safeguards, update that database. As evidenced by the high downtime, and the large fraction of that due to hardware failures, the system appears to be beset by maintenance problems. Over 100 hours of outage due to power and airconditioning failures are included in the 'other causes' classification.

Prime time is in this system defined as a ten hour interval between 8 am and 6 pm Eastern Time. For a subset of the equipment that includes the mainframes, separate failure statistics were kept for prime time and total time. These indicate that outages accounted for only 2.2% of the prime time compared to 3.66% of total time. While this again seems to contradict the workload dependence of failures, it is in this case due to an established policy which permitted shutdown of one of the redundant computers for maintenance during non-prime hours. Any failure in the active computer then propagated immediately to a system outage.

3.3 INTERPRETATION OF FINDINGS FOR FUTURE C3I SYSTEMS

The most encouraging reliability experience encountered in this survey was that reported for the No. 4 Electronic Switching System. The most prominent factors that account for the superior showing appear to be:

a. A large, dedicated development staff

b. Multiple installations of identical equipment

c. Extensive diagnostic programs for failure identification

d. Building on past experience with similar systems

The staffing practices at Bell Labs provide specialists in all aspects of reliability (from device physics through system architecture) within the project organization. Because of low employee turnover, individuals or small groups become highly expert at their assigned responsibilities. Factors (b) through (d) were also present to a large extent in the other systems for which multiple installations existed, and these probably account for lower downtime that was generally reported for these. None of the systems described on Table 3-1 had a downtime of more than 35 hours per year, whereas all but the airline systems described on Table 3-2 had downtime considerably in excess of 35 hours per year.

It is unlikely that the Government can procure C3I systems that have the legacy of development and operational experience inherent in a Bell Laboratories electronic switching system. Nonetheless, emphasis on thorough development and field testing prior to a committment to operation provides substantial reliability benefits and should be practiced. The other factors enumerated above are directly applicable to Government procurements and should be identified as requirements in future program plans. With proper attention to such requirements, it seems possible to achieve availability approaching 99.9% in a dual installation.

The analysis presented earlier in this section dwelt heavily on the workload dependence of computer failures because it is believed that this is particularly important for military C3I installations. In times of potential or real conflict, the workload in these systems is expected to increase very significantly, and it is under these circumstances that failures will have the worst effect. Thus, predicting an average availability of 99.9% for C3I equipment can be as misleading as stating that the average depth of a stream is one foot which leads to drowning of a party trying to ford that stream and finds that the maximum depth is much greater. The availability planning and prediction must be based on a stated workload that should reflect the maximum a given installation will be exposed to in case of military conflict.

Availability can be increased by furnishing additional spare resources in place, or by making remote spare resources accessible in case of a failure. Distributed systems have a high potential for facilitating the latter approach but here, again, an additional workload dependence needs to be recognized: utilization of remote computers requires high capacity data links, and these might be busy or unusable (EMI, etc.) during the time that they are needed to support geographically dispersed computing. These factors will be evaluated in later phases of this study.

The star configuration of networks is of particular interest in tactical military systems because it models the command structure. It was therefore significant to find that at least one network using that structure did not experience seriously adverse effects from the dependence of such a network on the continuous operability of a node.

SECTION 4 — DESIGN ISSUES AND METHODS IN DISTRIBUTED SYSTEMS

This section surveys previous work on both the identification of problems of distributed systems and design methods for their solutions. Subsection 4.1 provides a framework for describing and analyzing the wide variety of network design methodologies along with the problems and design issues they address. Subsection 4.2 discusses previous work relevant to C3I applications, and subsection 4.3 summarizes the results as a set of requirements for the design of fault tolerant distributed systems.

4.1. A FRAMEWORK FOR DESIGN METHODS AND ISSUES

This framework uses three descriptors to characterize design methodologies for distributed systems: design motivation, stage of network implementation, and scope. Design motivation refers to the attribute that is being optimized (e.g. cost, throughput, etc.). Stage of network implementation relates to the stage of development of networking components, and ranges from fully developed systems to network designs where neither the processors. links, terminals, or software have been developed. Scope is used to describe the range of design problems addressed from the formulation of requirements to the final detailed design.

4.1.1. Design Motivations

Motivations for distributed systems affect the approach to the design, the evaluation criteria used to make tradeoffs, and figures of merit used to assess performance. Past work on distributed systems can be classified on the basis of these motivations which include increasing throughput or response time, lowering communications costs, conforming to the structure of the user organization, relieving the load on an overburdened system, or increasing system reliability and availability.

Increasing system throughput or decreasing response times have been major motivations for the general research community, tactical C3I applications, real time control, and ballistic missile defense. The principal design issues have been optimization of task allocation with respect to throughput, efficient interprocess communication. distributed data bases. and, to a limited degree, fault tolerance (the ability to add or delete units in a distributed system provides this flexibility). Because such systems are generally both non-dispersed and under the control of a single local commander, distributed computing is not inherently superior to a central processor in these applications. However, because no hardware appropriate for field and battle conditions has the requisite throughputs. the use of several smaller units operating in parallel is a viable alternative.

Lowering communications costs has been of primary concern to both DoD and non-DoD agencies that operate general purpose computing facilities serving dispersed users. The major design issue is the tradeoff of the cost of local processing (e.g. for display formatting and local editing on CRTs) versus that of communication links with the capacity to transmit unreduced data to a central site for processing. Associated issues are the management and maintenance of a long-distance communications network, optimal task allocation with respect to cost (generally a nearly static proposition), minimization of system response time to user actions, and optimal choice of compatible hardware and software components. While high reliability is one of the design goals of these systems, fault tolerance is generally not. One significant exception, however, is the use of dual processor minicomputers (e.g. Tandem or Stratus) as front-end processors. Such systems are generally used on reservation or telephone ordering systems where high reliability is a requirement for marketing and customer relations.

Conforming to the structure of the user organization is also of concern to all classes of users. The primary C3I manifestation of this goal as the governing factor in distributed system design is evident in the structure of WWMCCS in which computing centers are associated with each of the major functions. Another, much smaller scale example, is the Xerox Ethernet based office automation network. The major issues are designing such a network in accordance with user and organizational requirements, providing for the configuration management and maintenance of a coherent network given the presence of heterogenous nodes, and failure diagnosis. Task allocation is not a consideration because nodes are generally not under the control of a central system supervisor. While communication costs. reliability, and network throughput must be within acceptable levels, these concerns are usually not as important as in networks motivated by the previous two considerations.

Relieving the load on an overburdened central computer is often a motivation for central computing facilities at major defense, scientific, and commercial sites. Generally, load relief involves installation of dedicated processors such as front end communication processors, interactive session processors, or back end data base machines. Major design issues are compatibility, throughput, and cost. An associated issue may be system reconfiguration and fault tolerance which is enabled by the presence of many interconnected computers. As was the case with the economically motivated system, task allocation is considered in the initial design, but will generally not occur dynamically unless automatic reconfiguration is provided as part of the fault tolérant implementation.

The final motivation, increasing reliability and availability, is of primary interest in the present study. The major design considerations include availability and effectiveness requirements, reconfiguration strategies (on the node, link, or system level), and acceptable degraded operating modes. System design requirements associated with throughput. performance, and cost define constraints for the highly reliable design. Most work on such systems has been performed in ar academic setting, and it has been concerned with relatively narrow issues (e.g. the number of nodes or link outages that can be tolerated in a switching network). This work has not considered reliability improvement as an integral part of a design that is primarily motivated by the issues previously discussed (increasing throughput, reduced communications cost, etc.). The integration of reliability enhancing design techniques — especially in the area of fault tolerance — into a general design methodology has not been adequately addressed; the second phase of the present investigation will be aimed at that area.

4.1.2. State of Network Development

For the purposes of this section, we consider four stages of system implementation:

(1) the network is already implemented, and the methodology deals with its interconnection with other networks

(2) network components (i.e. nodes, links, and software) and architecure have been developed, and the methodology deals with the optimal interconnection strategies

(3) computing nodes on the network have been developed and hardware interfaces are available, and the methodology deals with the interconnection and control of these computing resources

(4) no network components have been developed, and the methodology deals with general characteristics of networks.

The first stage is of importance to C3I systems on both the tactical and strategic levels. Examples include the interconnection of several tactical air defense C3I (e.g. AWACS and ground-based radar) systems into a single integrated tactical information center or the linking of radar detection sites (e.g. PAVE PAWS) with a central command site (e.g. NORAD). Literature on the design of networks in the first classification (i.e. networks of networks) centers on the concept of there being a single "gateway" that serves as an interface between networks. Because of the early stage of development of this concept, most of the published literature addresses compatibility issues, standards (e.g. ISO X.25), and the problems associated with getting such gateways to work at acceptable levels. Issues associated with fault tolerance, high throughput, or cost (beyond the minimum acceptable levels) are seldom treated in the literature.

The next stage, the design of networks around existing and (more or less) fully implemented architectures, has been treated in a number of design methodologies of various scopes (see next subsection). The issues addressed by these methodologies include the distribution of applications, placement of nodes, and choice of links (if several types are supported, e.g. telephone and dedicated lines). Design motivations include economics, performance, organizational, and reliability. Examples of commercially available architectures include IBM's SNA and Xerox's implementation of Ethernet.

The third classification is relevant to tactical C3I systems involving the interconnection of smaller individual computers and to strategic systems which may involve different types of processing performed on various machines (e.g. co-processors used together with an upgrade of the 427-M computers). Methodologies come in the form of articles and reports documenting the experience of researchers in constructing these networks. The methodologies deal with issues such as the design of the communication links, inter-computer protocols, operating system modifications, and those issues listed in the previous paragraph. Examples of "home-grown" networks include installations at Lawrence Livermore Labs and the China Lake Naval Weapons Center. Primary design motivations are related to increased throughput and relieving the load on existing mainframes.

The final classification is relevant to those C3I systems which are designed without the use of any developed computers. The motivations for such systems are increasing with the growing capabilities of microprocessors coupled with externally imposed constraints on weight, power consumption, and volume. Several methodologies have arisen from ballistic missile defense applications. All issues mentioned in previous paragraphs are relevant, and additional issues include the structure of the computer hardware, communication links, and the entire system software.

4.1.3. Scope of the Design Methodologies

An important characteristic of any design methodology is how much of the problem it covers. Nagle, et. al. [NAGL79] point out that design methodologies for fault tolerant distributed systems must begin at the system definition, or requirements level and proceed through to implementation. Different methodologies designate various steps in the design of computer-based systems; Figure 4-1, taken from Sloane and Wrobleski [SLOA82], shows that used at TRW. Many design methodologies in the literature do not address applications which are sufficiently specific that all the steps in Figure 4-1 are appropriate. Others have been developed for problems posed by the distributed system itself, not by any application induced requirements. Thus, for the purposes of this study, three general scope descriptors will be used:

Requirements-The methodology addresses the formulation and development of system requirements from functional or mission requirements stated in non-computer like terms.

Architecture - The methodology addresses specific issues in distributed systems design including protocols, task allocation, distributed operating systems, data bases, etc.

Communications - The methodology addresses issues related to choice of communication media, problems in their implementation, and monitoring of the network links.

4.2. PREVIOUS WORK

This section reviews some of the recent work on the identification of design issues and methodologies in distributed systems. Table 4-1 presents a grouping of the methodologies and a summary description based on the framework described in Subsection 4.1.

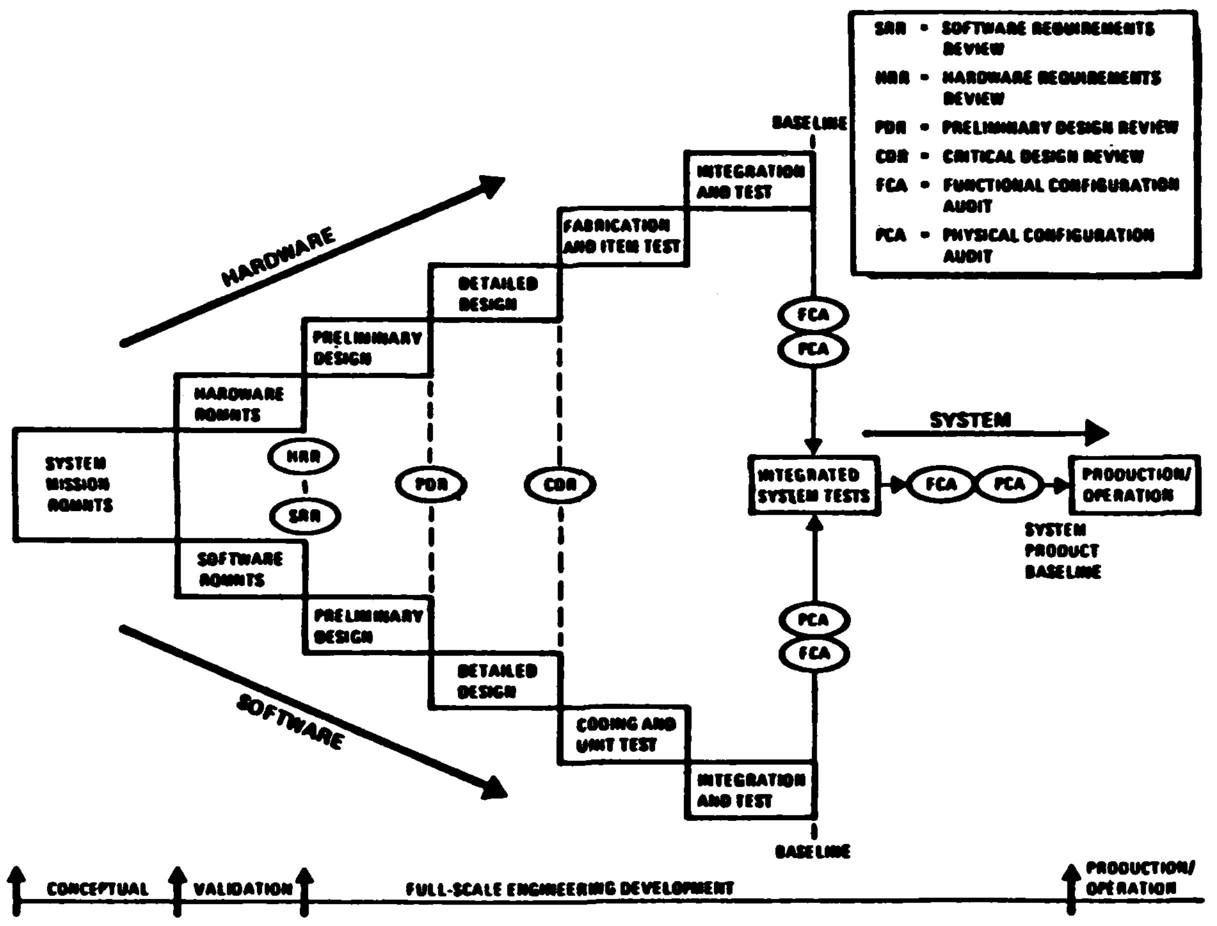


FIGURE 4 - 1 SYSTEM DESIGN METHODOLOGY AT TRW (FROM [SLOA82])

TABLE 4-1 DESIGN METHODOLOGIES AND RELATED TOPICS REVIEWED IN THIS SECTION

REFERENCE	APPLICATION	MOTIVATION	DEVELOPMENT	SCOPE
Alford [ALFO81]	BMD	Throughput Reliability	No components developed	Reqts. Arch.
Meier, Lemoine and Nam [MEIE81]	BMD	Throughput Reliability	No components developed	Arch.
FitzGerald & Eason [FITZ78]	Business	Economic Response time Reliability	Developed Sys. Arch.	Reqts.
Frankel [FRAN82]	Business	Economic	Developed Sys. Arch.	Commu.
DiCiccio, et. al. [DICI79]	Unspec.	Organizational (Inter-network)	Developed Sys. Arch.	Reqts.
Popek [POPE81]	Unspec.	Reliability	No components developed	Reqts.
Glen & Zimmerman [GIEN79]	Unspec.	Performance	Developed Network	Reqts.

4.2.1. Design of Distributed Systems in BMD Applications

Current design concepts for The Ballistic Missile Defense (BMD) systems emphasize local networks of computers. The primary design motivations are increased throughput, decreased response time, and improved availability. Such systems are generally not designed around any existing computers or networks, and thus, the entire range of distributed system design issues must be considered. Software issues include communications protocols, design of the distributed operating system (i.e. replicated and nonreplicated modules, interprocess communication, etc.), design and implementation of a distributed data base system, and task allocation.

Alford, et al. [ALFO81] have devised a distributed computing design system that is based on a methodology with eight top-level steps and a large number of lower level tasks and sub-tasks. The eight overall steps are system requirements definition, data processing (primarily in the area of operating systems, not communications) subsystem engineering, process design (i.e. defining and placing processing nodes and allocating computing tasks), sequential program design, code and unit test. Integration and test, and operation and maintenance. The design methodology is unique in clearly providing for the definition of critical functions, network reconfiguration, and alternate paths in the requirements phase and propagating these issues in the subsequent design steps. Its main contribution, however, is that it formalizes and structures the design process to the extent that many of the details, information interfaces, and error checking can be computerized.

Van Tilborg and Jasinksi [VANT81] deal with design issues in operating systems. The three major areas in the design of BMD distributed operating systems are interprocess communication (both within a node and between nodes), database management, and task allocation (during design, normal operation, and reconfiguration). The design objectives of interprocess communication protocols are (1) minimizing demands on processor throughput, (2) detecting, preventing, or avoiding deadlock, (3) reducing the amount of handshaking needed to synchronize the data exchange, and (4) ensuring that transmitted data is received undamaged. Issues in the design and operation of database systems include (1) where to put data bases with respect to the processors which will access them, (2) the extent to which the data should be replicated in order to reduce access times, and (3) how to minimize access times subject to database consistency and integrity requirements. Issues in task allocation include both distributing the tasks to the various nodes and scheduling them according to precedence and timing constraints.

Meier, Lemoine, and Nam [MEIE81] concentrate specifically on the issue of dynamic task allocation in an advanced Low Altitude BMD system. There are known task scheduling algorithms which can solve problems such as minimizing response time for a set of tasks subject to timing and precedence constraints. However, few of them are tractable for large systems. These authors evaluate computationally feasible (though not necessarily optimal) algorithms for effectiveness against specific threat scenarios by means of simulations. This approach can be quite useful in the development of reconfiguration and re-allocation schemes for fault tolerant systems.

4.2.2. Business Systems

Many large business oriented computer networks are quite similar to strategic C3I systems on all but the application specific level. Both environments use dispersed mainframe computer installations connected by a communications network, use similar communications hardware and software, and have similar reliability requirements. Thus, although requirements on the application level may differ somewhat, literature on the design and implementation of these systems is of relevance to this study.

The primary design motivation of distributed systems in business applications is the provision of an acceptable level of service at minimum cost and development time. As a result, such systems generally rely on commercially available networking systems which integrate hardware and software into a ready-made architecture that can be tailored to the requirements of the user. Examples of such products include IBM's System Network Architecture (SNA), initially designed for automated tellers, and Xerox's Ethernet, intended for office automation applications. Formulation of user requirements is the focus of most design methodologies in this area (of which there are a number); issues that may appear minor and subtle to the network designer can be major contributors to the success or failure of the network (e.g. placement of CRTs, consideration of power, space, and temperature requirements, etc.).

Requirements formulation for distributed systems in business applications must focus on three issues: (1) user requirements of network performance, (2) traffic that the network must bear, and (3) cost and time constraints. Options available to the system designer include CPUs, front end communication processors and PABXs, modems, tandem switching centers, multiplexers. concentrators, message switches, and common carrier services.

FitzGerald and Eason [FITZ78] define a ten-step procedure which can be grouped into three phases: pre-requirements, requirements, and implementation. The pre-requirements phase involves problem definition (in user terms), approach development, background information gathering on the organization, examination of the "people problems" and other associated issues affected by the distributed system, and generation of functional requirements (in user terms). The second phase consists of formulating system requirements and constraints, generating design alternatives that meet the requirements subject to the constraints, and choice of the best system. The implementation phase involves convincing management of the needs for the system, purchase and installation of the system, acceptance testing, development of operating and maintenance procedures, performance monitoring, and fine tuning.

This procedure differs from the previous BMD application in the following ways:

1. Because of the unwritten "organizational culture" with which the analyst may not be familiar, a large proportion of the requirements phase must be devoted to understanding not only explicit and quantifiable requirements but also implicit criteria which will affect the acceptability of the design.

2. The reluctance of most organizations to invest in distributed systems research and development necessitates the use of commercially available components with service and support from the vendor. Thus, most of the work in the development of design alternatives involves examination of the performance specifications and any credible reliability data of system components -- not on design of new devices.

3. The non-technical nature of the user organization requires special attention to "human factors" engineering in both the hardware and software, relations with the decision-making entities (i.e. management), and training beyond that required to operate specific software packages or systems.

System design issues are quite similar in other aspects. Certain load factors can be predicted (e.g. the transmission of administrative and financial information at predetermined intervals), while others can not (e.g. the interactive entry of customer orders). Concerns on the validity of transmitted information are often central for applications such as automated bank teller terminals just as they are for C3I applications. System availability and reliability for applications such as airline reservations are crucial to the economic well being (i.e. survival) of organizations just as they are in defense settings.

Frankel [FRAN82] concentrates on one aspect of system design -- topology of the communications network -- and on one criterion -- cost. Figure 4-2A shows six nodes connected to a center. In this example, the nodes are simple CRT terminals and the center is a minicomputer, but the considerations can be extended to any star network. Figure 4-2B shows the same functional configuration interconnected as a single multidrop line. The motivation for the multidrop configuration is cost: network A has a monthly cost twice that of network B at 1982 rates. However, other motivations may favor network a. For example, if link bandwidths are a constraint (as opposed to the processing capacity at the central node), or if the reliability of the links is low compared with that of other components, then A is preferable. On the other hand, if the multidrop link is a higher capacity line or consists of redundant paths, then such considerations would favor network B over A, although not at the same cost advantage.

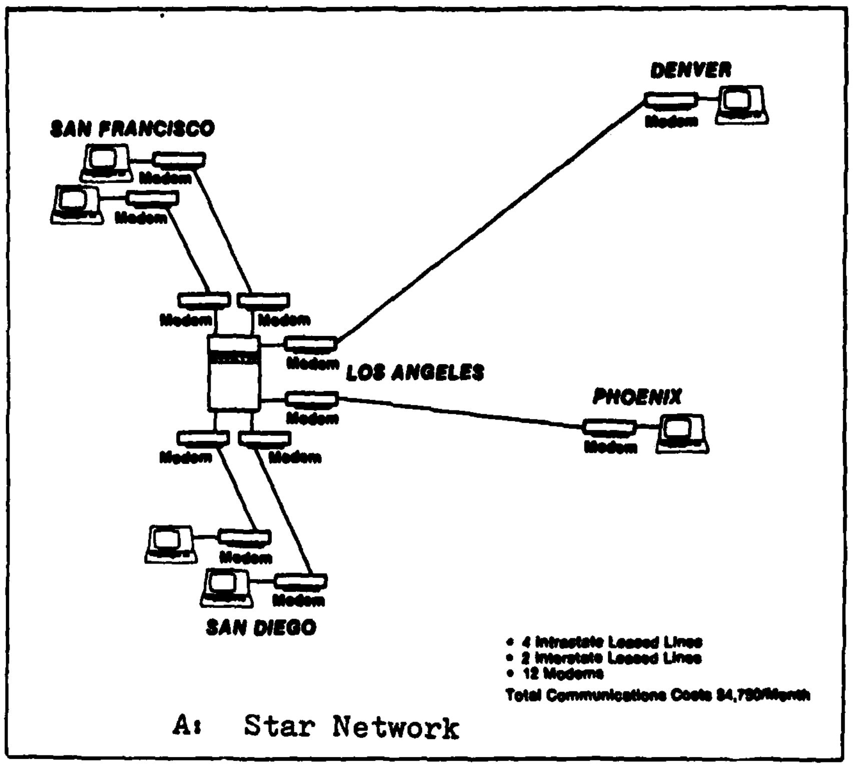
4.2.3. Networking Considerations

This subsection discusses design approaches and methodologies in terms of the network rather than in terms of an application. Major problems in this area include the interconnection of heterogeneous networks and computer systems as well as general software issues such as distributed operating systems or data bases.

Glen and Zimmerman [GIEN79] concentrate on the problems of network interconnection, and provide solutions in the form of analogies to heterogeneous computer networks, in which special interfaces must be provided. Figure 4-3 is a pictorial representation of the problem: given the fact that networks A and B are geographically dispersed and computationally incompatible, how do users X and Y communicate.
The primary design motivation is to ensure transparency to the end user.

In general terms, the method proposed by the two authors involves transferring the message from the user node to the network gateway (which may be a single processor or two "gateway halves", one located at each network), routing it through the gateway to the second network, and then passing it through the second network to the appropriate destination node. Such a strategy involves addressing (i.e. a local address for the gateway on network A, a global address designation on the gateway for network B, and a local address designation for node Y on network B), routing (through network A to the gateway, from the gateway to network B, and through network B to the destination), and the matching of incompatible protocols for error detection, flow control, and terminal control (by means of definition of a third protocol with interfaces to those of networks A and B).

The problem of interconnecting different networks through the installation of additional hardware and software interfaces and the implementation of additional layers in the communications protocol may minimize the impact on



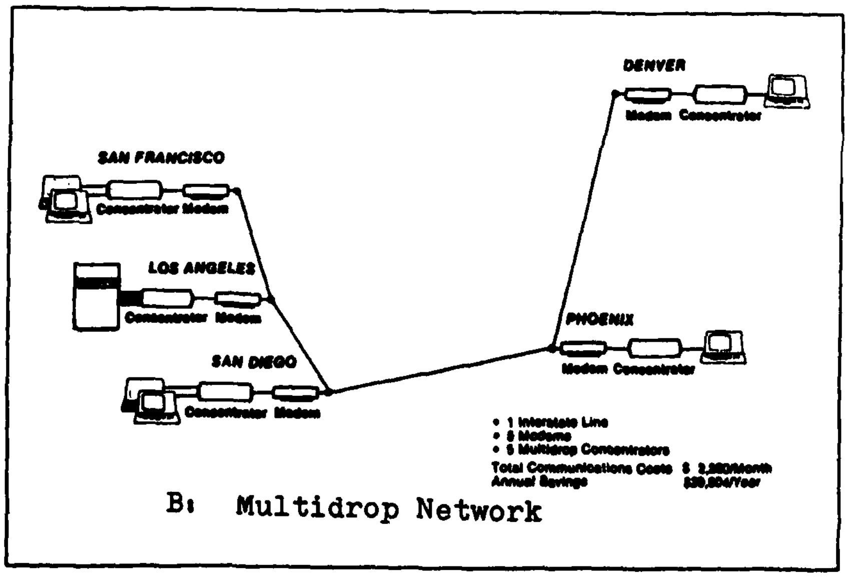


FIGURE 4 - 2 ECONOMICALLY MOTIVATED NETWORK DESIGN
(FROM [FRAN82])

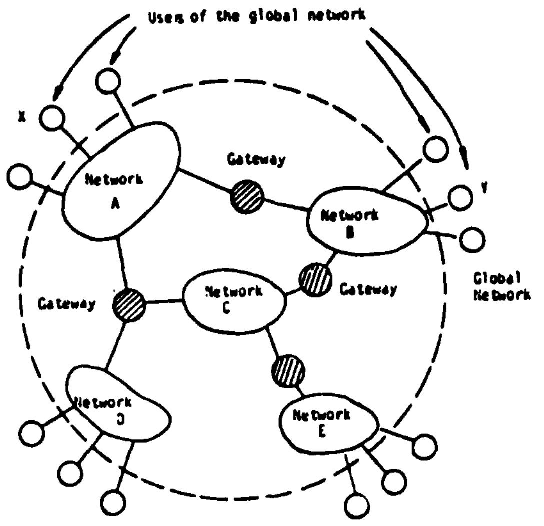


FIGURE 4 - 3 INTERCONNECTION OF HETEROGENEOUS NETWORKS (FROM [GIEN82])

existing systems but can cause reductions in throughput and reliability. For example, the presence of only single gateways between the networks poses significant reliability problems. However, if multiple gateways and network entry points are used, additional scheduling, addressing, and contention resolution issues have to be addressed. The additional complexity caused by *hierarchical addressing and routing schemes* can also result in reliability problems in both data integrity and correct execution of the protocols. Finally, the use of an intermediate protocol across the gateway further decreases throughput and reliability.

Alternate approaches are available. For example, the more closely the internal networks resemble each other, the less complex the interface. CCITT standard X.75 dictates some degree of internal network commonality [DICI79], and greater similarities can further reduce intercommunications problems. DiCicio, et. al. [DICI79] also discuss advantages to using packets rather than virtual circuits as a means of network interconnection for detecting a failure in the message cascade. Although their approach can lead to increases in throughput and error detection capability, it still contains drawbacks from the reliability — and especially from the fault tolerance — point of view.

Popek [POPE81] describes the reliability problems associated with a distributed data base. Partitioning is a means of preventing error propagation and is an important means of reducing the time necessary for restart and recovery. Redundancy is the means by which error detection occurs as well as a necessary part of any recovery process. Because distributed systems lend themselves to both partitioning and redundancy, they have considerable potential for highly reliable and available operation.

One of the major problems in distributed data bases is ensuring the integrity of the data in the event of a system crash. Three general techniques are available for this purpose: atomic transactions, two phase commit, and the transaction log. Atomic transactions are bracketed by "Begin Transaction" and "End Transaction" designations. In the event of a failure, it is the system's responsibility to ensure that all partially completed sequences of instructions are removed and all completed transactions are stored in the system's permanent memory. The two phase commit procedure involves a supervisor, a data sender, and a data receiver (all of which might be procedures resident on a single host). The supervisor queries both the transmitter and receiver on their status, and when both are ready, it commands the sender to transfer the data to the receiver. At the completion of the transfer, the supervisor commands the receiver to commit the transaction, and the receiver returns with a commit acknowledge signal. If the system crashes before the commit acknowledge, upon recovery, the system retains the previous value. Both the atomic action and two phase commit procedure require a transaction log in which intermediate values are stored and can be recalled in the event of a failure.

While such constructs are not unique to distributed data bases, their implementation over a slow and noisy network poses throughput and reliability problems. For example, the requirements of four messages in order to write an item to a non-resident data base may be unacceptable in many C3I applications. Thus, alternative techniques, examples of which are contained in the reference [POPE81], are necessary.

4.3 KEY ISSUES IN THE DESIGN OF FAULT TOLERANT DISTRIBUTED SYSTEMS

From the analysis of the design methodologies discussed above, certain key issues can be identified which will govern the design of fault tolerant C3I systems. Such systems must have the ability to:

1. Detect and identify failures on nodes and links
2. Re-establish contact to nodes in the event of a link failure by either (a) using an alternate link along the same path, or (b) establishing an alternate path.
3. Restore critical computer functions by either (a) reconfiguring the node to restore full capabilities on a local level or (b) re-allocating and scheduling tasks among other nodes.
4. Retain all critical data (and access to it)
5. Detect and recover from (or prevent) deadlock in the contention for resources, execution of tasks, or accessing of data.
6. Restore (or prevent) errors in data transmission and storage.
7. Access other critical networks in the event of a failure of the primary gateway.

SECTION 5 - FAULT LOCATION TECHNIQUES

The ability to locate (identify) faults is a key requirement in the implementation of fault tolerance. Most work in fault location has been carried out at the logic level [BREU76], and only a few authors have addressed fault location in networks of digital processors. Where the latter approach has been taken, as in [BLOU77], there has been emphasis on general applicability of the techniques rather than on specific implementations. To supplement that work, detailed fault location techniques for connected processors are described here on the basis of examples for specific configurations. All of the examples utilize a combination of pre-processors and mainframes with segmentation (switching provisions) between the pre-processors and the mainframes. The pre-processors may be signal processors, communication concentrators, or the gateway through which interactive processes are connected to the mainframe. The techniques described here are still applicable, with obvious simplifications, where no pre-processors are involved.

Three examples are treated here, all of them representative of configurations that were encountered in the study of existing fault tolerant or linked computer systems. Common assumptions and notation are discussed first. The subsequent headings in this section then describe fault location for

Single user segmented dual computer systems

Single user segmented dual computers with shared memory

Multiple user segmented dual computer systems

Fault location is defined as a process that is initiated after an error has been detected and after output devices that might be adversely affected by diagnostic procedures have been disconnected from the computers. Faults are assumed to be solid at the system level. This includes cases in which an internal transient fault has placed the computer into a state in which no further processing in accordance with requirements is possible.

5.1 ASSUMPTIONS AND NOTATION

A typical system of this type is shown in Figure 5-1, and the capital letter symbols used there are referred to in the following text.

5.1.1 <u>Assumptions</u>

(1) Test initiation and evaluation. In order to locate the source of the fault, there must be at least one accessible reliable component that can then test other components adjacent to it and thereby create a directory of functioning components. The following sequence of operations will be followed in each test. First, the user selects (randomly, if necessary) a reliable component and initiates a prestored self-test routine. If this fails, another

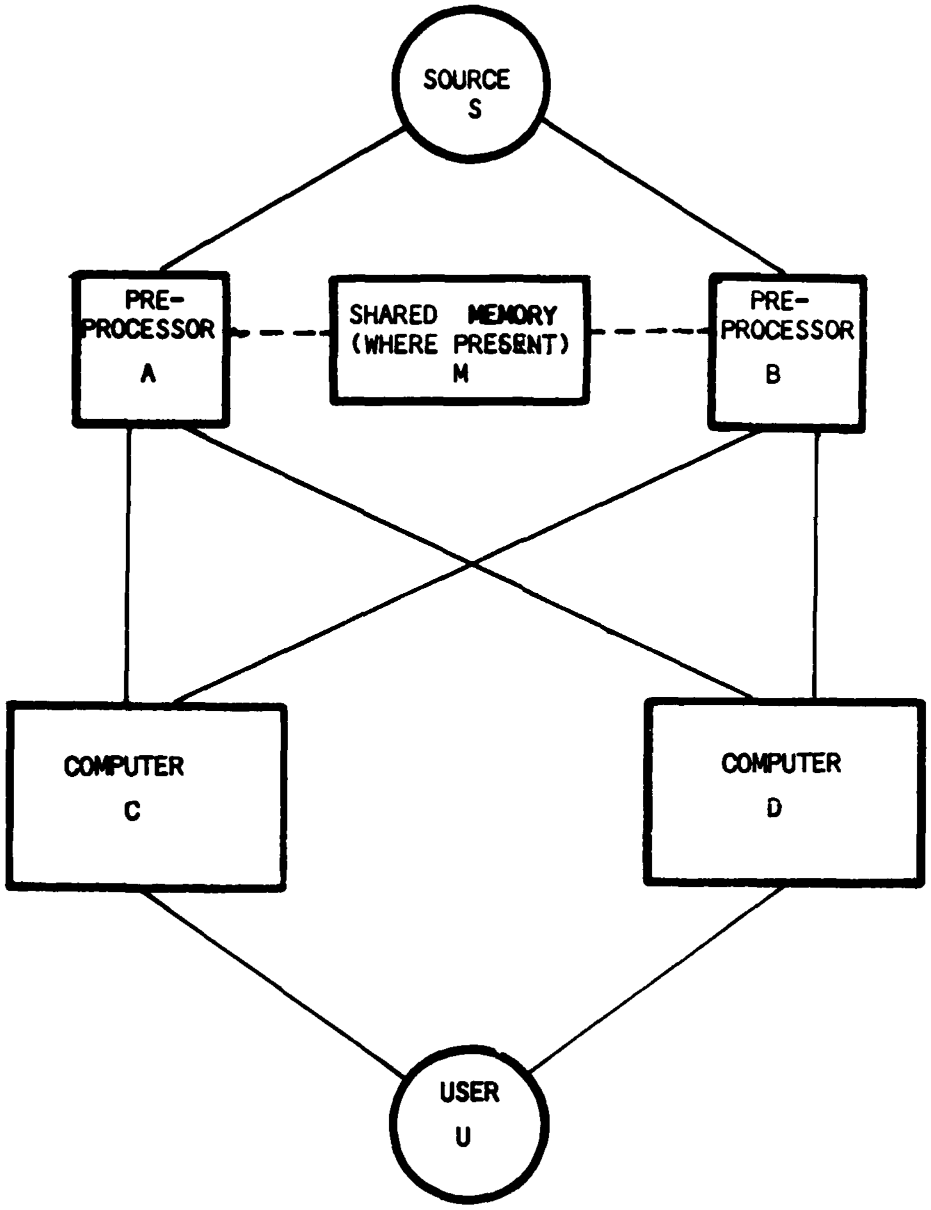


FIGURE 5 - 1 SINGLE USER SEGMENTED SYSTEM

component is selected. The first reliable component identified by this process then stimulates another unit under test (UUT) to execute a predefined diagnostic routine, and it expects to receive the results generated by this routine. If no results are returned within a specified time, the UUT is marked as malfunctioning. If results are returned, the reliable component compares them with a stored benchmark and accepts the UUT as operative only if all results agree.

(2) User involvement. As a baseline for the fault location procedures, a substantial amount of user involvement has been assumed. While the sequence of units to be tested is identified in the procedures presented below, the actual issuance of commands to implement the sequence is assumed to be performed by the user. In principle it is possible to store the sequence and issue it as a single command. However, the conditions encountered in the early part of the test affect the actions to be taken in later ones. Recognition of these conditions, which may involve the interpretation of outputs generated by a malfunctioning computer, is in general best handled by a trained human observer, possibly with the aid of some computer functions. The performance of fully automated diagnostics for an unrestricted fault set on arbitrary computer architectures is a specialized research area outside the scope of the effort reported on here. Likewise, in the baseline approach, the user is expected to select an appropriate repair or reconfiguration action after the fault condition has been identified by the procedures described here. Certain sequences in the procedures are arbitrary, e. g., whether to start the test with processor C or D in Figure 5-1. In order to generate a repeatable procedure, processor C was selected as the first UUT. A knowledgeable user may decide on the basis of past history or immediate observations that D is more likely to be at fault and therefore start the test there. These deviations are considered permitted but they are not an essential part of the user involvement in the test procedures.

(3) Perfect test coverage. Generally, the time and storage cost for a test is proportional to the thoroughness of the testing of a hardware component. It was assumed that sufficient resources can be allocated for a test that gives a very high assurance that it will not pass a malfunctioning component (nearly 100% test coverage). The failure of software used for testing was not allowed for. To the extent that actual diagnostics do not yield 100% test coverage, a malfunctioning component might be declared operable and a failure will then occur in a later test step or during operation. User involvement can alter the sequence of testing so that another operable combination of components can be configured.

(4) Two-way transmission. It was assumed that links can carry test-related messages in both directions. The bandwidth required for this purpose is small because the stimulus is usually expressed as a single command, and the result can be compressed.

(5) Distinction between processors and links. It is often difficult to draw a clear distinction between failures in a link and in a processor connected to that link. If a malfunction disables processor P's communication with all its neighbours Q1 .... Qn, then the failure is attributed to the processor although it might be a common failure in all links. On the other hand, if the

failure leaves at least one communication path between P and QI operable, then the failure is attributed to the affected links although it might be a failure in the processor that affects a portion of the communications capabilities.

(6) Irrecoverable faults. The purpose of locating faulty processors is to remove them from the net and to resume real-time operations. However, there must be at least one normally functioning path between the input (S in Figure 5-1) and the user (U). Faults which do not leave such a path are not worth locating because the system can not be automatically restored to useful service. The fault location procedures therefore stop as soon as an irrecoverable fault has been identified.

(7) Preprocessors with shared memory (applicable to 5.3 only). A test of a preprocessor involves use of shared memory and therefore tests the shared memory. It is assumed that the shared memory has error correcting code that masks transient and isolated permanent memory faults. Therefore only solid failures affecting substantial areas of the memory will affect preprocessor operation. Memory is regarded as functioning if at least one preprocessor passes tests involving shared memory. Links to the shared memory are treated as part of the preprocessors served by them since a preprocessor without access to shared memory is not suitable for normal operation.

5.1.2 <u>Notation and Types of Tests</u>

Three types of tests are used in the fault location procedures:

Type 1 - Direct user test

Type 2 - Test with only forward information flow

Type 3 - Test with reverse information flow

Examples of the notation used and of the application of these tests are given below.

Type 1 - notation U -t-> C

This denotes a test in which the user (U) stimulates computer C and receives results from it. This test is applicable only to processors directly accessible by the user, such as C or D in Figure 5-1.

Type 2 - notation C -t-> A

This type of test is used for establishing operability of the preprocessors and associated links. It is assumed that the selection of the tester (C) and of the UUT (A) is made by the user, and that the user has visibility of the outcome (at least pass/fail) of the test.

Type 3 - notation Ad -t-> C

This means that A, while being stimulated through D, tests C (backward flow of information). The lower case letter is used in lieu of a subscript. If U -t-> C has failed while U -t-> D and D -t-> A have succeeded, it is not clear whether C has failed or whether the link from the user to C is inoperable. By sending the test initiation order through D for A to test C, an independent means is found for determining whether C is operable.

Two symbols connected by a dash represent the link between the elements designated by the symbols, e. g., U-D stands for the link between the user and computer D, and A-C stands for the link between computers A and C.

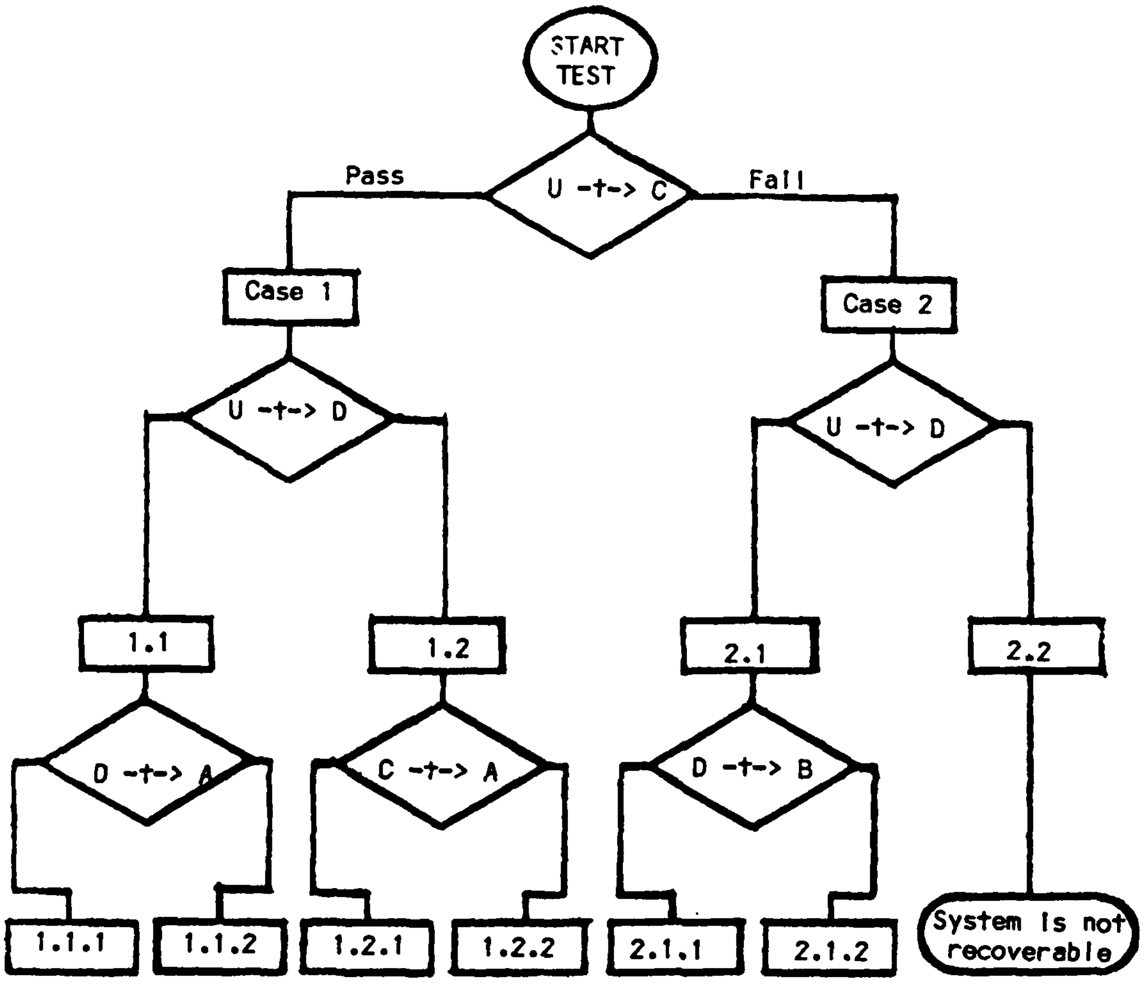
5.2 SINGLE USER SEGMENTED DUAL COMPUTER SYSTEMS

The fault location procedure presented below applies to the single user segmented system without shared memory. The structure connected by the broken line in Figure 5-1 is not present for this case.

The procedure consists of first testing the processors connected to the user interface, C and D (and, by implication, the links to the user). After these have been found to be operational, the processors at the source interface, A and B, and their backward links (to C and D) are tested. The links from A and B to the source are considered to be part of the latter and are not explicitly validated in this procedure. If the preprocessors check out on the test described here and yet no useable information is received in the operational mode, failure of the source links or of the source is implied.

The normal, forward directed (upward in Figure 5-1), part of the test is flowcharted in Figures 5-2 through 5-4. Where failures were encountered in tests initiated directly by the user (of the form U -t-> X), backward tracing is used in later phases of the test to determine whether the failure affects an entire computer or only the user link or interface. Certain other links are also diagnosed separately from the processors which they serve by means of backward directed tests. These diagnostics are shown in Figures 5-5 through 5-8. Not all computer installations that use the strucuture of Figure 5-1 may have the capability to perform backward directed tests. This capability is not essential for a determination of the operational status (i. e., which processors are accessible and working properly), but where it is not provided many link failures can not be distinguished from processor failures.

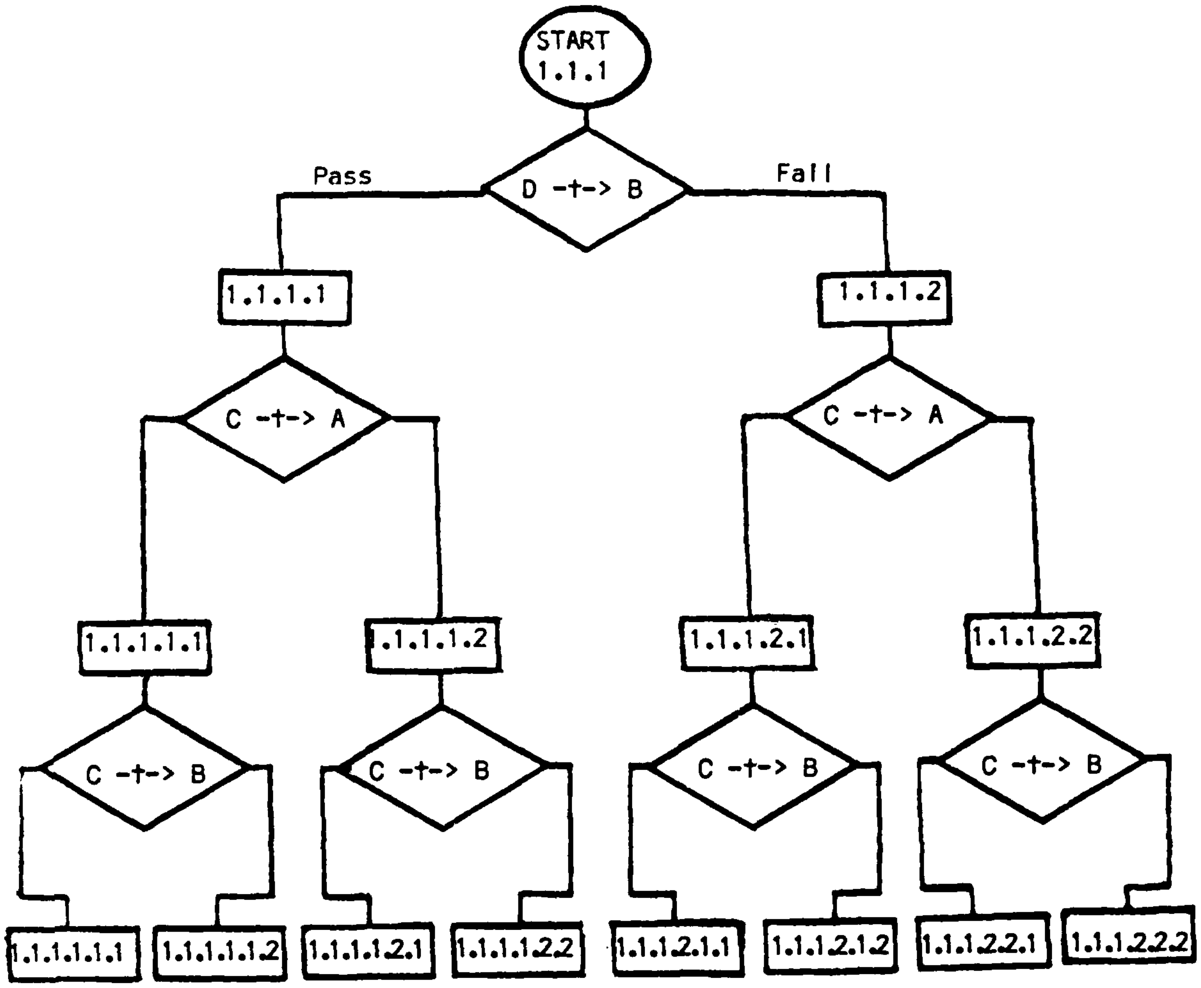
A summary of the diagnostic information obtained at each step of the fault location procedure is shown in the lower part of each figure. The case number is the number sequence shown in the rectangular boxes after each test. The designation 'tested' used in the summary means that the components or links were identified as operational in the sequence performed up to this point. In some instances a computer can be identified as operational in the test sequence although it can not be accessed in normal operation due to failures in other processors or links. These computers are designated as 'non-usable' in the diagnostic summaries (single asterisk). In other cases, the diagnostics can not distinguish between processor failures or simultaneous failures in all links to a processor; this situation is identified by a double asterisk in the summaries. From the operational point of view, it makes no difference whether the processor or all links have failed.



SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
1	None	C	1
1.1	None	C,D	2
1.1.1	None	A,C,D	3
1.1.2	A or A-D	C,D	2
1.2	D or U-D	C	1
1.2.1	D or U-D	A,C	2
1.2.2	D or U-D, A or A-C	C	1
2	C or U-C	none	0
2.1	C or U-C	D	1
2.1.1	C or U-C	B,D	2
2.1.2	C or U-C, B or B-D	D	1
2.2	C or U-C, D or U-D	none	0

FIGURE 5 - 2 FAULT LOCATION PROCEDURE OF SECTION 5.2, PART 1

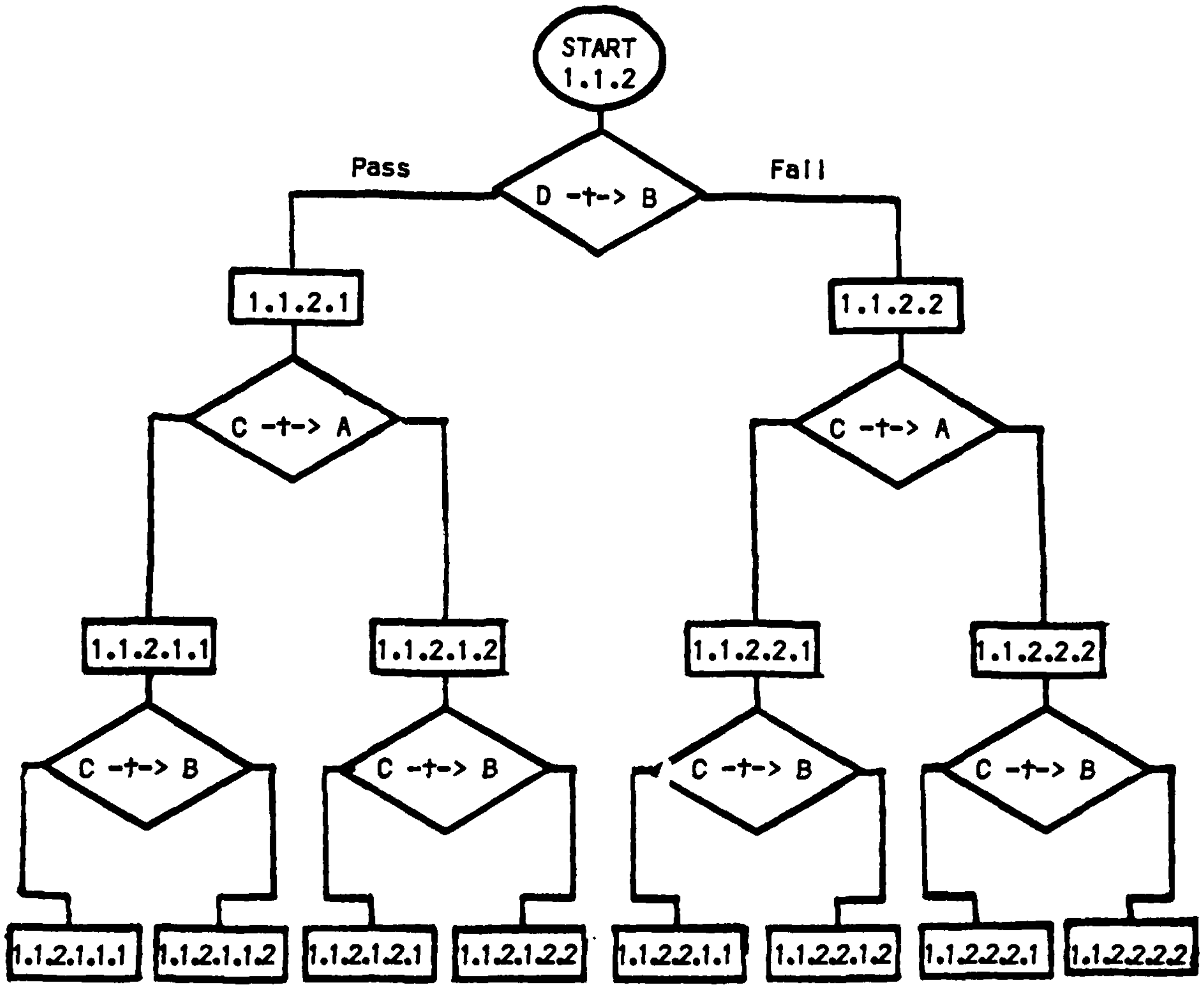


SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
1.1.1	None	A, C, D	3
1.1.1.1	None	A, B, C, D	4
1.1.1.1.1	None	A, B, C, D	5
1.1.1.1.1.1	None	A, B, C, D	6
1.1.1.1.1.2	B-C	A, B, C, D	5
1.1.1.1.2	A-C	A, B, C, D	4
1.1.1.1.2.1	A-C	A, B, C, D	5
1.1.1.1.2.2	A-C, B-C	A, B, C*, D	4
1.1.1.2	B or B-D	A, C, D	3
1.1.1.2.1	B or B-D	A, C, D	4
1.1.1.2.1.1	B-D	A, B, C, D	5
1.1.1.2.1.2	B**	A, C, D	4
1.1.1.2.2	B or B-D, A-C	A, C, D	3
1.1.1.2.2.1	B-D, A-C	A, B, C, D	4
1.1.1.2.2.2	B**, A-C	A, C*, D	3

* non-usable ** processor or all connections have failed

FIGURE 5 - 3 FAULT LOCATION PROCEDURE OF SECTION 5.2, PART 2

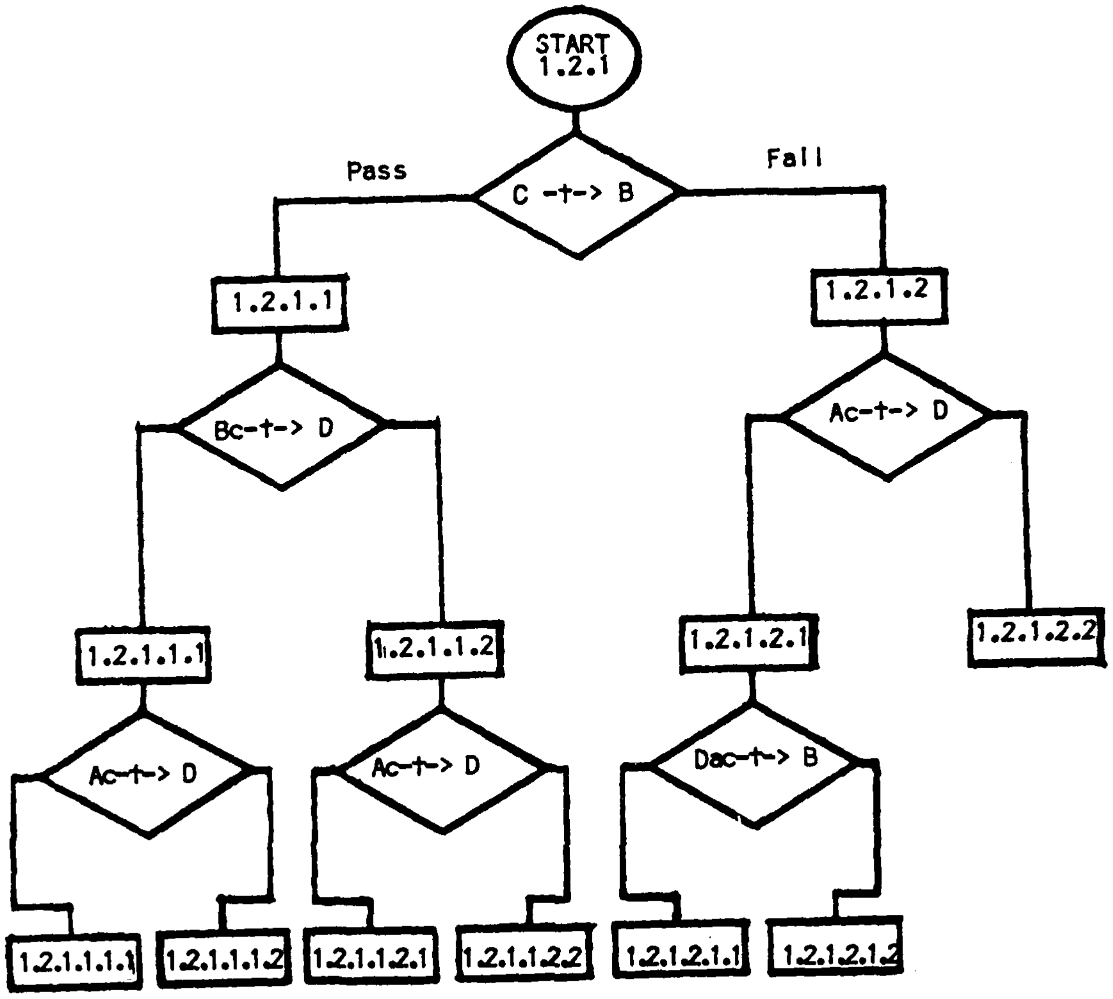


SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
1.1.2	A or A-D	C, D	2
1.1.2.1	A or A-D	B, C, D	3
1.1.2.1.1	A-D	A, B, C, D	4
1.1.2.1.1.1	A-D	A, B, C, D	5
1.1.2.1.1.2	A-D, B-C	A, B, C, D	4
1.1.2.1.2	A**	B, C, D	3
1.1.2.1.2.1	A**	B, C, D	4
1.1.2.1.2.2	A**, B-C	B, C*, D	3
1.1.2.2	A or A-D, B or B-D	C, D*	2
1.1.2.2.1	A-D, B or B-D	A, C, D*	3
1.1.2.2.1.1	A-D, B-D	A, B, C, D*	4
1.1.2.2.1.2	A-D, B**	A, C, D*	3
1.1.2.2.2	A**, B or B-D	C, D*	2
1.1.2.2.2.1	A**, B-D	B, C, D*	3
1.1.2.2.2.2	A**, B**	C, D* (not oper.)	2

* non-usable ** processor or all connections have failed

FIGURE 5 - 4 FAULT LOCATION PROCEDURE OF SECTION 5.2, PART 3

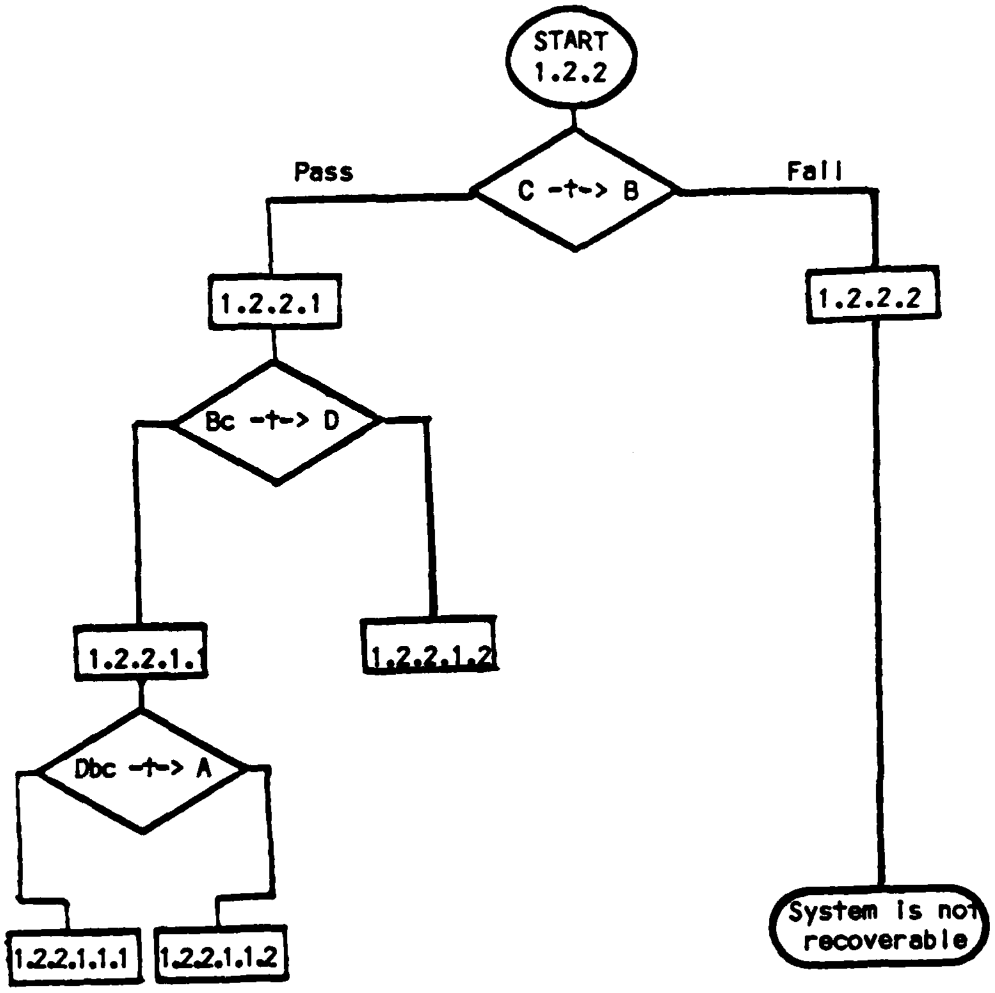


SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
1.2.1	D or U-D	A, C	2
1.2.1.1	D or U-D	A, B, C	3
1.2.1.1.1	U-D	A, B, C, D*	4
1.2.1.1.1.1	U-D	A, B, C, D*	5
1.2.1.1.1.2	U-D, A-D	A, B, C, D*	4
1.2.1.1.2	D or (U-D & B-D)	A, B, C	3
1.2.1.1.2.1	U-D, B-D	A, B, C, D*	4
1.2.1.1.2.2	D**	A, B, C	3
1.2.1.2	D or U-D, B or B-C	A, C	2
1.2.1.2.1	U-D, B or B-C	A, C, D*	3
1.2.1.2.1.1	U-D, B-C	A, B*, C, D*	4
1.2.1.2.1.2	U-D, B**	A, C, D*	3
1.2.1.2.2	B**, D**	A, C	2

* non-usable ** processor or all connections have failed

FIGURE 5 - 5 FAULT LOCATION PROCEDURE OF SECTION 5.2, PART 4

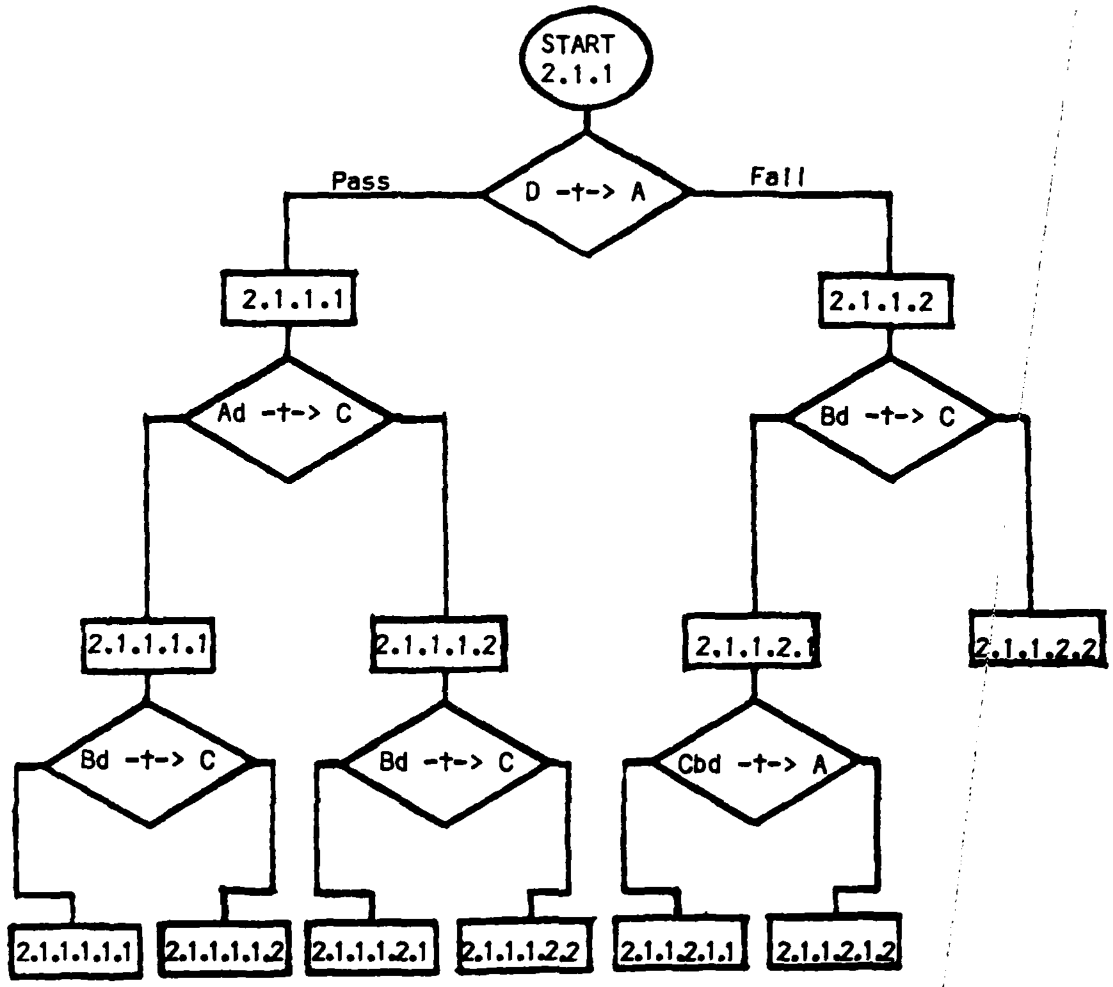


SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
1.2.2	D or U-D, A or A-C	C	1
1.2.2.1	D or U-D, A or A-C	B, C	2
1.2.2.1.1	U-D, A or A-C	B, C, D*	3
1.2.2.1.1.1	U-D, A-C	A*, B, C, D*	4
1.2.2.1.1.2	U-D, A**	B, C, D*	3
1.2.2.1.2	D or (U-D&B-D), A or A-C	B, C	2
1.2.2.2	D or U-D, A or A-C, B or B-C	C* (non-oper.)	1

* non-usable ** processor or all connections have failed

FIGURE 5 - 6 FAULT LOCATION PROCEDURE OF SECTION 5.2, PART 5

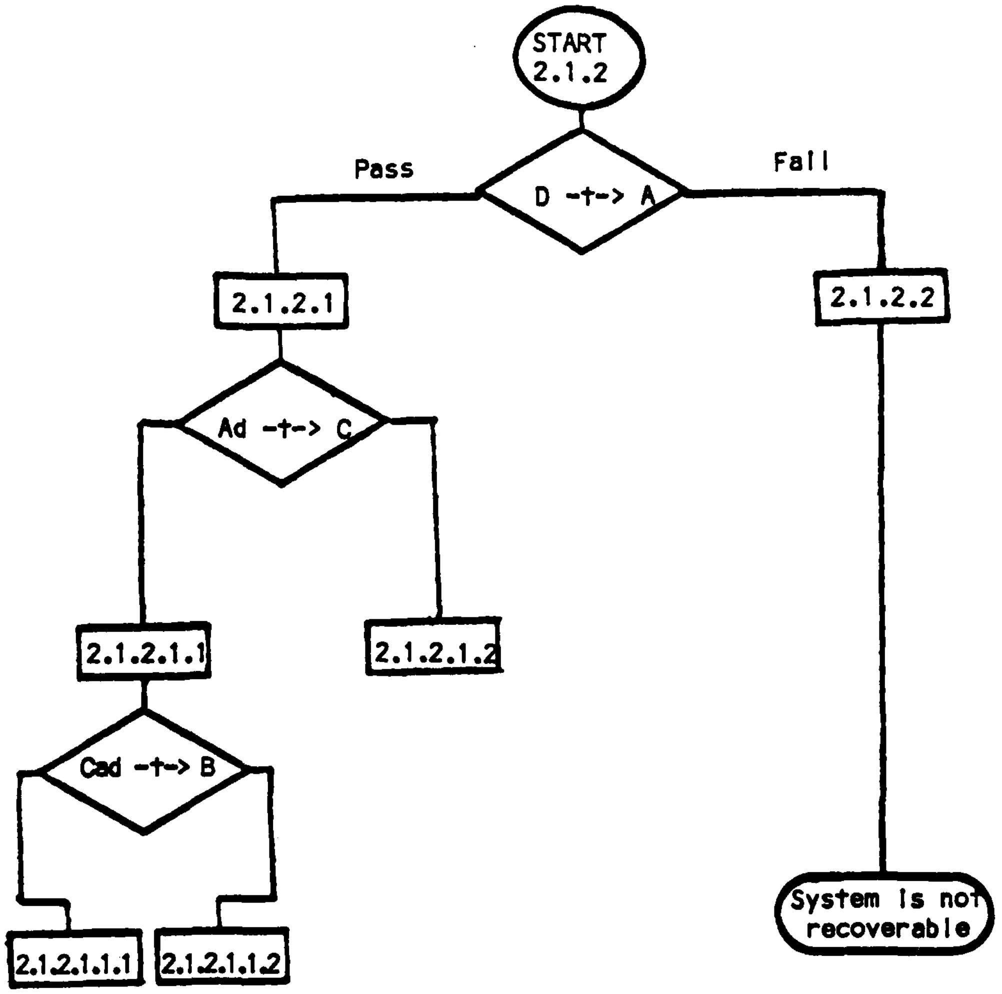


SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
2.1.1	C or U-C	B, D	2
2.1.1.1	C or U-C	A, B, D	3
2.1.1.1.1	U-C	A, B, C*, D	4
2.1.1.1.1.1	U-C	A, B, C*, D	5
2.1.1.1.1.2	U-C, B-C	A, B, C*, D	4
2.1.1.1.2	C or (U-C & A-C)	A, B, D	3
2.1.1.1.2.1	U-C, A-C	A, B, C*, D	4
2.1.1.1.2.2	C**	A, B, D	3
2.1.1.2	C or U-C, A or A-D	B, D	2
2.1.1.2.1	U-C, A or A-D	B, C*, D	3
2.1.1.2.1.1	U-C, A-D	A*, B, C*, D	4
2.1.1.2.1.2	U-C, A**	B, C*, D	3
2.1.1.2.2	A**, C**	B, D	2

* non-usable ** processor or all connections have failed

FIGURE 5 - 7 FAULT LOCATION PRODECURE OF SECTION 5.2, PART 6



SUMMARY OF DIAGNOSTICS

Case	Failed Components	Tested Components	No. of Tested Links
2.1.2	C or U-C, B or B-D	D	1
2.1.2.1	C or U-C, B or B-D	A, D	2
2.1.2.1.1	U-C, B or B-D	A, C*, D	3
2.1.2.1.1.1	U-C, B-D	A, B*, C*, D	4
2.1.2.1.1.2	U-C, B**	A, C*, D	3
2.1.2.1.2	C or (U-C&A-C), B or B-D	A, D	2
2.1.2.2	C or U-C, A or A-D, B or B-D	D* (non-oper.)	1

* non-usable   ** processor or all connections have failed

FIGURE 5 - 8 FAULT LOCATION PRODECURE OF SECTION 5.2, PART 7

The detailed procedure for performing these tests is presented in a design language based on Pascal in the Appendix. Where case designations are used in the appendix, they correspond to those shown in the flowcharts; however, not all case designations shown on the flowcharts are mentioned in the design language version of the test procedure.

5.3 SINGLE USER DUAL COMPUTERS WITH SHARED MEMORY

The fault location procedure presented under 5.2 above is also effective for the case where the two processors have shared memory (the broken line in Figure 5-1 represents the connection). The analysis presented below interprets the outcomes of the procedures of 5.2 for the case of shared memory.

It is assumed that any failure in the shared memory will result in failure of tests for both Processor A and Processor B. Therefore, if either Processor A or Processor B is found to be operational it may be assumed that the shared memory is functioning correctly. Conversely, when both processors are found to be inoperative there is a high probability that the shared memory has failed, although this case cannot be distinguished by the gross diagnostics used here from a simultaneous failure of A and B or from a failure of all backward connections (lines going down in Figure 5-1). The detailed test data will usually permit differentiation between processor and memory failures.

The diagnostics furnished by the tests shown in Figures 5-2 through 5-8 are analyzed in Table 5-1. If a test case results in a definitive finding regarding the shared memory (either usable or not usable , this finding will also be valid for all subsidiary test cases, and they are not separately listed. Thus, the finding that the shared memory is usable for 1.1.1 implies that it is also usable for all cases 1.1.1.x.x.x where x may represent either a 1 or a 2. The figure numbers shown in the table are valid until a new figure number is shown.

A finding of 'not usable ' for the shared memory can arise either from inability to access either one of the processors using the memory, or from the observation that both are inoperative. In the former case, the status of the memory is really unknown, and this is indicated by a single x in the table. In the latter case, it is highly likely that the shared memory has failed, although, as indicated above, other possibilities cannot be completely ruled out, and this case is designated by xx in the table.

TABLE 5 - 1 SHARED MEMORY DIAGNOSTICS

Case	Shared Memory Usable	Shared Memory Not useable	Further Diagnostics Required	Ref. Figure No.
1.1.1	x			5-2
1.1.2			x	
1.2.1	x			
1.2.2			x	
2.1.1	x			
2.1.2			x	
2.2		x		
1.1.2.1	x			5-4
1.1.2.2			x	
1.1.2.2.1	x			
1.1.2.2.2			x	
1.1.2.2.2.1	x			
1.1.2.2.2.2		xx		
1.2.2.1	x			5-6
1.2.2.2		xx		
2.1.2.1	x			5-8
2.1.2.2		xx		

5.4 MULTIPLE USER SEGMENTED COMPUTER SYSTEMS.

A typical configuration of this type is shown in Figure 5-9. It will be recognized that this figure is identical with Figure 5-1 except for the connections at the user and source ends. To capitalize on the similarity, it is convenient to divide the fault location procedure into three phases that establish the operability of (1) the user interface, (2) the computer network proper and (3) the source interface. Phase 1 and Phase 3 procedures are developed in detail below. Phase 2 procedures represent an adaptation of those described in Section 5.2.

At the start ot Phase 1, each user must determine the accessibility of computers C and D by a sign-on procedure. When this is completed, there will be an access log within C and D which will be of the form (U1)(U2)(U3) where each term will have a value of 1 if Ui has logged in and a value of 0 otherwise. Thus, if the access log for computer C is 101, this means that users U1 and U3 can access this computer and user U2 cannot. Any 0 value represents a diagnostic for an inoperative user link. In addition, the ensemble of the access logs determines the procedure to be followed in Phase 2. For that purpose, the outcomes of Phase 1 can be classified in the following manner:

1a. One or more users can access both C and D

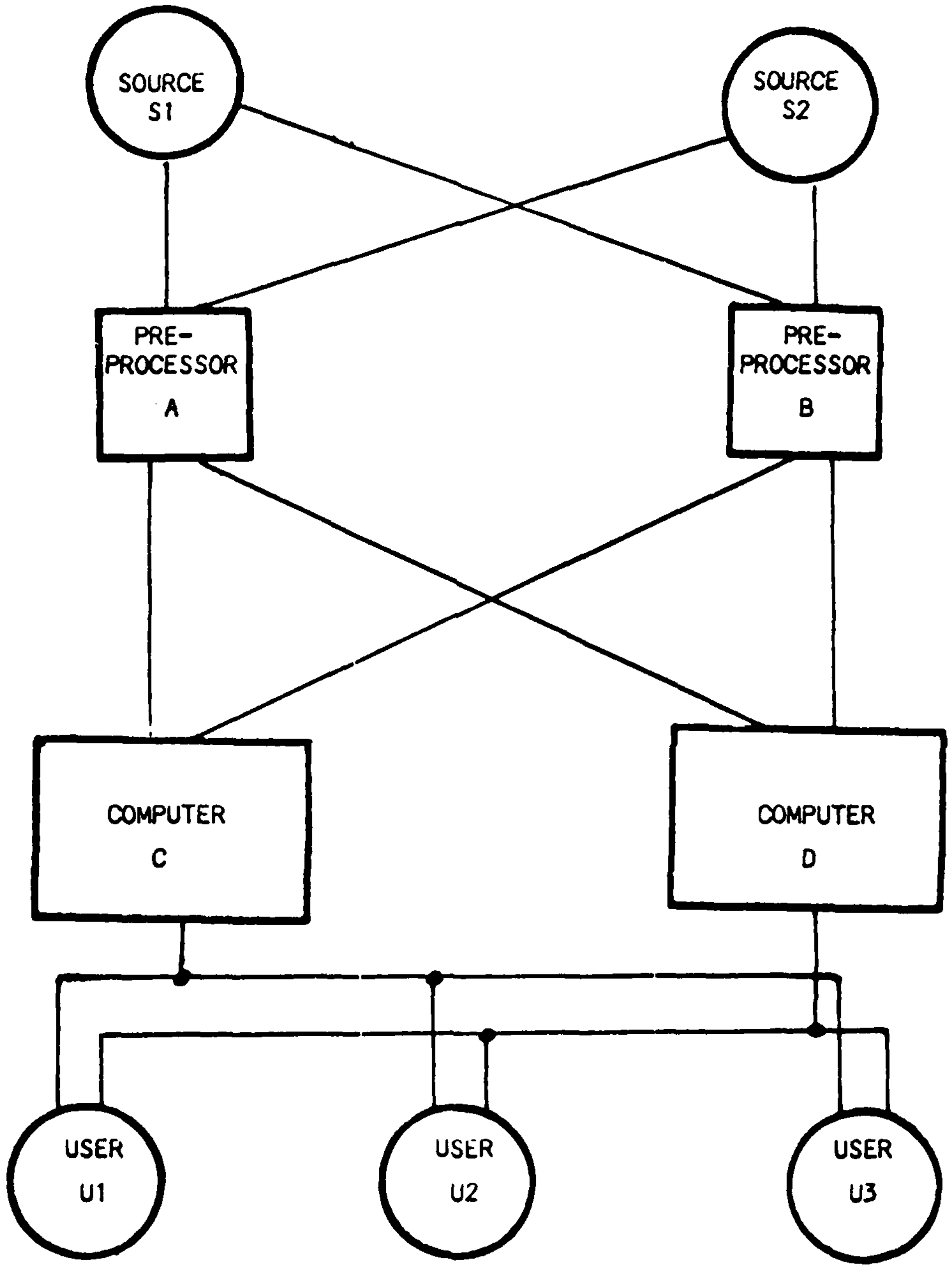


FIGURE 5 - 9 MULTIPLE USER SEGMENTED SYSTEM

1b. C and D can both be accessed, but not by the same user(s)

1c1. C can be accessed by one or more users, D cannot be accessed

1c2. D can be accessed by one or more users, C cannot be accessed

1d. Neither C nor D can be accessed by any user.

The classification of Phase 1 outcomes is derived from the access log codes generated within the C and D computers as shown in Table 5-2. The "1" prefixes have been omitted in the table.

TABLE 5 - 2 CLASSIFICATION OF PHASE 1 OUTCOMES

Access	Access Log D							
Log C	111	110	101	011	100	010	001	000
111	a	a	a	a	a	a	a	c1
110	a	a	a	a	a	a	b	c1
101	a	a	a	a	a	b	a	c1
011	a	a	a	a	b	a	a	c1
100	a	a	a	b	a	b	b	c1
010	a	a	b	a	b	a	b	c1
001	a	b	a	a	b	b	a	c1
000	c2	c2	c2	c2	c2	c2	c2	d

If Phase 1 produces an outcome in the 1a. classification, Phase 2 can be initiated by any user who can access both computers, and the procedure of Section 5.2 can be applied without modification. If Phase 1 produces an outcome in the 1b. classification, separate actions by two users will be necessary during the Phase 2 procedure. The user who can access C (but not D) proceeds in accordance with case 1.2 on Figure 5-2, and the user who can access D (but not C) proceeds in accordance with case 2.1 in Figure 5-2. If the Phase 1 procedure results in a 1c1. classifcation, only the case 1.2 procedure can be initiated, and if it results in a 1c2. classification only the case 2.1 procedure can be initiated. Where Phase 1 terminates with a 1d. classification the system is not recoverable, and no Phase 2 activity can be conducted.

A similar classsification of Phase 2 outcomes is utilized to determine the Phase 3 procedure. These classifications are based on the operability of the A and B computers (these are also sometimes referred to as preprocessors) in accordance with Table 5-3.

TABLE 5 - 3 CLASSIFICATION OF PHASE 2 OUTCOMES

Computer	Computer B	
A	Operable	Not Operable
Operable	2a	2b1
Not Operable	2b2	2c

With both preprocessors operative (case 2a), the fault location technique can distinguish between a source failure and a failure of a single link associated with a sensor. When only one of the preprocessors is operative (cases 2b1 and 2b2), this distinction can not be made. When both preprocessors are inoperative, no diagnostics of the source subsystem are possible.

Testing of the sources and links in Phase 3 involves observation by the A and B computers of predefined characteristics of the input data stream, such as frequency of bit value transitions, frequency of start of cycle characters and absence of alarm characters. The observations at each processor are in the following designated as (S1)(S2) where a value of 1 for S1 designates an operable condition (predefined characteristics are present), and a value of 0 designates an inoperable condition. Thus, if the observation at computer A has a value of 10 this means that source S1 appears operable and source S2 appears inoperable. The classification of the combined observations from computers A and B (for Phase 2 outcome of 2a) during Phase 3 is shown in Table 5-4.

TABLE 5 - 4 CLASSIFICATION OF PHASE 3 OUTCOMES

Computer	Computer B			
A	11	10	01	00
11	3a	3b	3b	3c
10	3b	3d	3bb	3e
01	3b	3bb	3d	3e
00	3c	3e	3e	3f

These classifications have the following meaning

3a. Both sources fully usable

3b. One source fully usable ; one source usable on one link only

3bb. Both sources usable on one link only

3c. Both sources accessible from only one preprocessor

3e. Only one source accessible from one preprocessor

3f. No sources accessible

For a Phase 2 outcome of 2b1. only the last column in Table 5-3 is applicable, and for a Phase 2 outcome of 2b2. only the last row in Table 5-3 is applicable. A Phase 2 outcome of 2c is indistinguishable from a Phase 3 outcome of 3f.

APPENDIX

DESIGN OF A FAULT LOCATION PROGRAM

A computer program for the fault location procedure described in Section 5 has been designed. The specific fault location procedures are listed here in a Pascal-like design language. [[and]] are used as shorthand notations for "Begin" and "End", respectively. < Routine name > indicates transfer to a routine that is listed later.

A.1 SINGLE USER SEGMENTED DUAL COMPUTER SYSTEMS

Reference Figure 5-1. The shared memory is not present in this case.

```
procedure locate;
case (U -t-> C) of
pass: "case 1: C & (C-U) are ok"
   case (U -t-> D) of

   pass: "case 1.1: D & (D-U) are ok"
      case (D -t-> A) of
      pass: "case 1.1.1: A & (A-D) are ok"
         <locate-1.1.1>;
      fail: "case 1.1.2: A or (A-D) is malfunctioning"
         <locate-1.1.2>
      end "case 1.1";

   fail: "case 1.2: D or (D-U) is malfunctioning"
      case (C -t-> A) of
      pass: "case 1.2.1: A & (A-C) are ok"
         <locate-1.2.1>;
      fail: "case 1.2.2: A or (A-C) is malfunctioning"
         <locate-1.2.2>
      end"case-1.2"
   end"case-1;

fail: "case 2: C or (C-U) is malfunctioning"
   case (U -t-> D) of
   pass: "case 2.1: D & (D-U) are ok"
      --- This case is similar to case 1.2 except that ---
         --- C is interchanged with D and A with B ---
   fail:
            --- The system is not recoverable ---
end"locate"
```

```
procedure locate-1.1.1;
case (D -t-> B) of

pass: "case 1.1.1.1: B & (B-D) are ok"
 [[if (C -t-> A) then mark '(A-C) is ok'
                 else mark '(A-C) is malfunctioning';
   if (C -t-> B) then mark '(B-C) is ok'
                 else mark '(B-C) is malfunctioning']];

fail: "case 1.1.1.2: B or (B-D) is malfunctioning"
 [[if (C -t-> A) then mark '(A-C) is ok'
                 else mark '(A-C) is malfunctioning';
   if (C -t-> B) then mark 'B & (B-C) are ok and
                         (B-D) is malfunctioning'
                 else mark 'B is malfunctioning or
                            (B-C) & (B-D) are malfunctioning']]

end"locate-1.1.1".

procedure locate-1.1.2;
case (D -t-> B) of

pass: "case 1.1.2.1: B & (B-D) are ok"
 [[if (C -t-> A) then mark 'A & (A-C) are ok and
                            (A-D) is malfunctioning'
                 else mark 'A is malfunctioning or
                            (A-C) & (A-D) are malfunctioning';
   if (C -t-> B) then mark '(B-C) is ok"
                 else mark '(B-C) is malfunctioning']];

fail: "case 1.1.2.2: B or (B-D) is malfunctioning"
 [[if (C -t-> A) then mark 'A & (A-C) are ok and
                            (A-D) is malfunctioning'
                 else mark 'A is malfunctioning or
                            (A-C) & (A-D) are malfunctioning';
   if (C -t-> B) then mark 'B & (B-C) are ok and
                            (B-D) is malfunctioning'
                 else mark 'B is malfunctioning or
                            (B-C) & (B-D) are malfunctioning']]

end"locate-1.1.2".

procedure locate-1.2.1;
case (C -t-> B) of

pass: "case 1.2.1.1: B & (B-C) are ok"
   case (Bc -t-> D) of
     pass:  "case 1.2.1.1.1: D & (B-D) are ok and
                             (D-U) is malfunctioning"
       if (Ac -t-> D) then mark '(A-D) is ok'
                      else mark '(A-D) is malfunctioning';
```

```
      fail: "case 1.2.1.1.2: D is malfunctioning or
                             (B-D) & (D-U) are malfunctioning"
         if (Ac -t-> D) then mark 'D & (A-D) are ok and
                             (B-D) & (D-U) are malfunctioning'
                        else mark 'D or (A-D) is malfunctioning'
   end"case-1.2.1.1";

fail: "case 1.2.1.2: B or (B-C) is malfunctioning and
                     both are unusable and (B-D) is also unusable"
   case (Ac -t-> D) of
   pass: "case 1.2.1.2.1: D & (A-D) are ok and
                          (D-U) is malfunctioning"
        if (Dac -t-> B) then mark 'B & (B-D) are ok and
                          (B-C) is malfunctioning.
                          D, (A-D), B, and (B-D) are unusable'
                        else mark 'B is malfunctioning or
                                  (B-C) & (B-D) are malfunctioning';

   fail: "case 1.2.1.2.2"
        mark 'D is malfunctioning or (A-D) & (D-U) are malfunctioning.
              B, D, (B-C), (A-D) and (B-D) are unusuable.'
   end"case 1.2.1.2"

end"locate-1.2.1"

procedure locate-1.2.2
case (C -t-> B) of

pass: "case 1.2.2.1: B & (B-C) are ok"
   case (Bc -t-> D) of
   pass: "case 1.2.2.1.1: D & (B-D) are ok and
                  (D-U) is malfunctioning"
        if (Dbc -t-> A) then mark 'A & (A-D) are ok and
                  (A-C) is malfunctioning.
                  (A-D), A, and (A-C) are unusuable;
                        else mark 'A is malfunctioning or
                                  (A-C) & (A-D) are malfunctioning';
   fail: "case 1.2.2.1.2"
   mark 'D is malfunctioning or (B-D) & (D-U) are malfunctioning'

   end "case 1.2.2.1";

fail:  "case 1.2.2.2: B or (B-C) is malfunctioning"
                --- The system is not recoverable ---


end "locate-1.2.2"
```

A.2 SINGLE USER DUAL COMPUTERS WITH SHARED MEMORY

The fault location procedure presented under A.1 above is also effective for the case where the two processors have shared memory (the broken line connection in Figure 5-1 is present). The notes and procedures presented below interpret the outcomes of the procedures of A.1 for the case of shared memory.

(1) "case 1.1.1": Preprocessor A passed a test and thus it is reasonable to conclude that shared memory M is ok.

(2) "case 1.1.2.1": M is ok.

(3) "case 1.1.2.2":

```
[[if (C -t-> A) then mark 'A & (A-C) are ok and
   (A-D) is malfunctioning and
   M is ok'
 else mark 'A is malfunctioning or
   (A-C) & (A-D) are malfunctioning';
  if (C -t-> B) then mark 'B & (B-C) are ok and
   (B-D) is malfunctioning and
   M is ok'
 else [[mark 'B is malfunction!ng or
     (B-C) & (B-D) are malfunctioning']];
  if M has not been validated then mark 'M may be malfunctioning']]
```

(4) "case 1.2.1": M is ok.

(5) "case 1.2.2.1": M is ok.

(6) "case 1.2.2.2": M may be malfunctioning.

(7) "case 2.1": This case is the same as case 1.2 except for exchanging C with D and A with B.

(8) "case 2.2": (U -t-> C) = (U -t-> D) = fail": M's status is unknown.

A.3 MULTIPLE USER SEGMENTED COMPUTER SYSTEMS.

Reference Figure 5-9. The fault location procedure consists of three phases that are described in the following. Additional notations introduced are

MS - The set of elements identified as malfunctioning
WS - The set of elements validated as working

```
procedure phase1; "identification of usable main processors"

case ((U1 -t-> C),(U2 -t-> C),(U3 -t-> C)) of
(pass,pass,pass): "WS = [C,(C-U1),(C-U2),(C-U3)]"
                  mark 'C, (C-U1), (C-U2), & (C-U3) are ok';
```

```
(pass,pass,fail): "WS = [C,(C-U1),(C-U2)]; MS = [(C-U3)]"
                  mark 'C, (C-U1), & (C-U2) are ok and
                        (C-U3) is malfunctioning';
(pass,fail,pass): "WS = [C,(C-U1),(C-U3)]; MS = [(C-U2)]"
                  mark 'C, (C-U1), & (C-U3) are ok and
                        (C-U2) is malfunctioning';
(pass,fail,fail): "WS = [C,(C-U1)]; MS = [(C-U2),(C-U3)]"
                  mark 'C & (C-U1) are ok and
                        (C-U2) & (C-U3) are malfunctioning';
(fail,pass,pass): "WS = [C,(C-U2),(C-U3)]; MS = [(C-U1)]"
                  mark 'C, (C-U2), & (C-U3) are ok and
                        (C-U1) is malfunctioning';
(fail,pass,fail): "WS = [C,(C-U2)]; MS = [(C-U1),(C-U3)]"
                  mark 'C & (C-U2) are ok and
                        (C-U1) & (C-U3) are malfunctioning';
(fail,fail,pass): "WS = [C,(C-U3)]; MS = [(C-U1),(C-U2)]"
                  mark 'C & (C-U3) are ok and
                        (C-U1) & (C-U2) are malfunctioning';
(fail,fail,fail): "MS = [C or [(C-U1),(C-U2),(C-U3)]]"
                  mark 'C or [(C-U1),(C-U2),(C-U3)] is malfunctioning'
end"case";

case ((U1 -t-> D),(U2 -t-> D), ---
    --- same as above except that
       (1) C is replaced by D,
       (2) WS = [---] is replaced by WSnew = WSold + [---], and
       (3) MS = [---] is replaced by MSnew = MSold + [---].  ---

end"case"

end"phase1".
```

If neither C nor D can be used by any user, then the system is irrecoverable and the fault location procedure stops.

The actions taken during Phase 2 are chosen on the basis of the results of Phase 1. The possible results of Phase 1 can be classified into the following cases:

<u>case</u> 1.a: Both C and D can be used by the same user.
e.g., Both (U1 -t-> C) and (U1 -t-> D) resulted in "pass".

<u>case</u> 1.b: There is no user who can use both C and D but C can be used by one user Ui and D can be used by another user Uj.
e.g., ((U1 -t-> C),(U1 -t-> D)) resulted in "(pass,fail)" while ((U2 -t-> C),(U2 -t-> D)) resulted in "(fail,pass)".

<u>case</u> 1.c: Only one main processor, C or D, can be used by any user.
This can be divided into two subcases.
<u>case</u> 1.c.1: C is usable but D cannot be used by any user.
<u>case</u> 1.c.2: D is usable but C cannot be used by any user.

case 1.d: None of the main processors are usable.

In the last case (case 1.d), there is no Phase 2 actions. In other cases, the actions taken in Phase 2 are the same as some parts of the fault location procedure described in Section 5.2. The details of the Phase 2 actions are as follows:

```
procedure phase2; "Diagnosis of processors"

case results-of-phase1 of

1.a:  "This corresponds to case 1.1 in the fault location procedure in Section
      5.2. Using the same procedure, the operability of processors and their
      interconnections can be obtained.";

1.b:  "This also corresponds to case 1.1 in the fault location procedure in
      Section 5.2 except that whenever C needs to communicate with a user,
      e.g., in the case of C -t-> A, Ui is involved, whereas whenever D needs
      to communicate with a user, Uj is involved.";

1.c.1:"This corresponds to case 1.2 in the fault location procedure in Section
      5.2. Using the same procedure, the operability of processors and their
      interconnections can be obtained.";

1.c.2:"This corresponds to case 2.1 in the fault location procedure in Section
      5.2. Using the same procedure, the operability of processors and their
      interconnections can be obtained";

1.d:  "The system is irrecoverable." stop

end"case"

end "Phase 2".
```

The actions taken during Phase 3 depend on the results of Phase 2. These are classified into the following cases.

case 2.a: Both preprocessors are usable.

case 2.b: Only one preprocessor is usable.

case 2.b.1: A is usable but B cannot be used by any user.

case 2.b.2: B is usable but A cannot be used by any user.

case 2.c: None of the preprocessors are usable.

Here it is assumed that a preprocessor can tell the operability of a source by watching if readable information comes from the source. Therefore, A -t-> S1 means that A observes the information coming from source S1 and then makes a report on the status of S1.

The details of Phase 3 are as follows:

```
procedure phase3; "Diagnosis of sources"

case results-of-phase2 of

2.a: "A & B are usable"
   [[case ((A -t-> S1),(B -t-> S1)) of
     (pass,pass): mark 'S1, (S1-A), & (S1-B) are ok';
     (pass,fail): mark 'S1 & (S1-A) are ok and
                        (S1-B) is malfunctioning';
     (fail,pass): mark 'S1 & (S1-B) are ok and
                        (S1-A) is malfunctioning';
     (fail,fail): mark 'S1 or [(S1-A),(S1-B)] is malfunctioning'
     end"case";

     case ((A -t-> S2), ---
        --- same as above except that S1 is replaced by S2 ---
     end"case"]];

2.b.1: "A is usable"
   [[case (A -t-> S1) of

     pass: mark 'S1 & (S1-A) are ok';
     fail: mark 'S1 or (S1-A) is malfunctioning'
     end"case";

     case (A -t-> S2) of
     pass: mark 'S2 & (S2-A) are ok';
     fail: mark 'S2 or (S2-A) is malfunctioning'
     end"case"]];

2.b.2: "B is usable"
   [[ --- same as in case 2.b.1 except that A is replaced by B --- ]];

2.c: "The system is irrecoverable." stop

end"case"

end"phase3".
```

A.4 EXTENSIONS OF THE TECHNIQUES

A.4.1 Fault Location in a Reduced Configuration

After a malfunctioning component has been located, the component is functionally removed from the system and the rest of the system continues to operate. The removal of the component is recorded in the system status table. If another fault is detected later, a slightly modified version of the fault location procedure described earlier is followed.

The modification is that in each component the test is preceded by an examination of the system status table to determine whether the component in question has been functionally removed. The component test will follow only if the component has not been removed, and then the previously described procedures will be used.

A.4.2 Diagnostic Information Contained in Reports from Fault Detectors

It is possible to skip certain steps in the fault location procedure described in Section 5.4 by exploitingthe information contained in the reports made by fault detectors. There are two types of components capable of detecting faults: preprocessor and main processor. A preprocessor is capable of telling whether a source is functioning or dead. On the other hand, a main processor may be capable of telling if a preprocessor is dead or not. Moreover, if the two preprocessors have been assigned to process the same data, then a main processor should be able to detect a mismatch between the outputs of the two preprocessors. In all these cases, a preprocessor which detected a fault should send a report to all the users.

case 1: Preprocessor Y reported "Source SI is dead".

There are six possible paths from a preprocessor to the users.

case 1.1: The other preprocessor Y' made the same report.
conclusion: Source SI is indeed dead.

case 1.2: Preprocessor Y' did not make the report.
conclusion: Link (SI-Y), preprocessor Y', or all the paths from Y' to users are malfunctioning.

case 1.3: The report from Y did not come through all six paths.
conclusion: There are malfunctioning components on those paths which the report did not come through.

case 2: Main processor Z reported "Preprocessor Y is dead".
There are three links from a main processor to the users.

case 2.1: The other main processor Z' made the same report.
conclusion: Preprocessor Y is indeed dead.

case 2.2: Main processor Z' did not make the report.
conclusion: Link (Y-Z), main processor Z', or all the links from Z' to users are malfunctioning.

case 2.3: The report from Z did not come through all three links.
conclusion: The links which the report did not come through are malfunctioning.

case 3: Main processor Z reported "The outputs of the two preprocessors disagree".

case 3.1: The other main processor Z' made the same report.
conclusion: At least one of the preprocessors is malfunctioning.

case 3.2: Main processor Z' did not make the report.
conclusion: Main processor Z' or all the links from Z' to users are malfunctioning.

case 3.3: The report from Z did not come through all three links.
conclusion: The links which the report did not come through are malfunctioning.

The knowledge obtained as above can be used to shorten to a certain extent the fault location procedure to follow. However, it will increase the complexity of the overall fault location procedure and will not change the worst-case execution time of the location procedure. Therefore, the decision on whether to exploit the information contained in the fault report or not should be made with a consideration of the operational mode of the network (e.g., dual redundant operation, concurrent processing of different data, etc.), time constraints, logical complexity constraints, etc.

The fault location procedures described earlier do not distinguish a malfunctioning processor from an isolated processor. For example, failures of (A-D), (B-C), and (D-U) will make B, D, and (B-D) useless. Such finer resolution as distinguishing a malfunctioning processor from an isolated processor cannot be obtained without adding some links to the system. However, such additional information is useful primarily for the maintenance action (whether to repair or a processor) rather than for fault tolerance and run-time reconfiguration.

REFERENCES

ALFO81 M. Alford, et. al., Distributed Computer Design System, TRW Report No. 38392-6950-003, Ballistic Missile Defense Advanced Technology Center Contract No. DASG60-81-C-0059, August, 1981

ANDE75 G.A. Anderson and E.D. Jensen. "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", ACM Computing Surveys, vol. 7, no. 4, December, 1975

BANN81 J.A. Bannister, K Trivedi, V. Adlakha, and T.A. Alspaugh, Problems Related to the Integration of Fault Tolerant Aircraft Electronic Systems, Research Triangle Institute Report No. RTI/2094/02-02F, August, 1981

BEAU79 M. Danielle Beaudry, "A statistical analysis of failures in the SLAC computing center", Digest, COMPCON'79, pp. 49-52, IEEE cat. no. 79CH1393-8C, February 1979

BLOU77 Marion L. Blount, "Probabilistic treatment of diagnosis in digital systems", Proceedings, FTCS-7, pp 72-77, IEEE Cat No. 77CH1223-7C, June 1977

BREU76 M. A. Breuer and A. D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, 1976

CAST81 X. Castillo and D. P. Siewiorek, "Workload, performance, and reliability of digital computing systems", Digest, FTCS-11, pp. 84-89, IEEE cat. no. 81CH1600-6, June 1981

CORR79 J.P. Corr and D.H. Neal, "SNA and Emerging International Standards", IBM Systems Journal, vol. 18, no. 2. p. 244, 1979

DAVI81 E. A. Davis and P. K. Giloth, "No. 4 ESS: performance objectives and service experience", Bell System Technical Journal, vol. 60, no. 6, pp. 1203-1224, August, 1981

DICI79 V. DiCiccio, C.A. Sunshine, J.A. Field, and E.G. Manning, "Alternatives for Interconnection of Public Packet Switching Data Networks", Proc. 6th Data Communications Symposium, pp. 120-125, IEEE, 1979

ECKH78 R. H. Eckhouse and J.A. Stankovic, "Issues in Distributed Processing -- An Overview of Two Workshops", Computer, January, 1978, pp. 22-26

ELME72 W.R. Elmendorf, "Fault Tolerant Programming", Digest of Papers, 1972 International Symposium on Fault Tolerant Computing, IEEE Catalog No. 72CH0623-9C, June, 1972

ENSL78 P.H. Enslow, Jr., "What is 'Distributed Data Processing'", Computer, January, 1978, pp. 13-21

ENSL81 P.H. Enslow, Jr., "Distributed Data Processing -- What is It?", AGARD Conference on Tactical Airborne Distributed Computing and Networks", AGARD CPP 303, June, 1981

FILA82 Rome Air Development Center/ DCLW, "Flexible Intraconnect Local Area Network (FILAN) Specification", MIL-STD-(FI) USAF, August, 1982

FITZ78 J. FitzGerald and T.S. Eason, Fundementals of Data Communication, Wiley, 1978

FRAN82 J. Frankel, "Mainframe Data Communications for Minicomputers", Datamation, March, 1982, p. 178

GAO78 U.S. General Accounting Office, "NORAD's Information Processing Improvement Program -- Will It Enhance Mission Capability?", LCD-78-117, September, 1978

GIEN79 M. Gien and H. Zimmerman, "Design Problems for Network Interconnection", Proc. Sixth Data Communications Symposium, IEEE, pp. 109-119, November, 1979

GRAY80 W. D. Gray, "FAA's en route air traffic control computer system", Investigations Staff, Committee on Appropriations, U. S. Senate, Report 80-5, October 1980

GREE77 Wm. Greene and U.W. Pooch, "A Review of Classification Schemes of Computer Communication Networks", Computer, November, 1977

ISO81 International Standards Organization, "Open Systems Interconnection", Draft Standard TC-97, August, 1981

IYER82 R. K. Iyer and D. J. Rosetti, "A statistical load dependency model for CPU errors at SLAC", Digest, FTCS-12, pp. 363-372, IEEE cat. no. 82CH1760-8, June 1982

JENS76 E.D. Jensen, K.J. Thurber, G.M. Schneider, "A Review of Systematic Methods in Distributed Processor Interconnection", Proceedings, ICC '76, reprinted in Tutorial: Distributed Processor Communications Architecture, IEEE Cat. No. EHO 152-9, 1979

KEIS64 W. Keister et al., "No. 1 electronic switching system", Bell System Technical Journal, vol. 43, pt. 1 and 2, September 1964

LARI81 S.J. Larimer and L. Walker, "A Continuous Reconfiguring Multi-Microprocessor Flight Control System", Wright Patterson Air Force Base AFWL-TR-81-3070, May, 1981

MEIE81 L. Meier, A.L. Lemoine, and C.W. Nam, Decentralized Control for Ballistic Missile Defense Systems, Systems Control Inc. Report 5266, BMD System Command Contract No. DASG-60-78-C-0069, June, 1981

NAGL79 H.T. Nagle, et. al., Fault Tolerance and Architectural Reliability in Distributed Data Processing Systems, Auburn University Report No. AU-EE-0025-1, May, 1979, p. 1-4

POPE81 G.J. Popek, "Notes on Distributed Systems of Microprocessors", in G. Goos, J. Hartmans. eds, Lecture Notes in Computer Science: Microcomputer Systems Design, Springer Verlag, Berlin, 1981

SHOC82 J.F. Shoch, et. al., "Evolution of the Ethernet Local Computer Network", Computer, vol. 15, no. 8, p. 10, August, 1982

SLOA82 E. Sloane and J. Wrobleski, "Software Product Assurance: Learning Lessons from Hardware", presentation to joint meeting of Los Angeles Chapter IEEE Computer Society and Reliability Society, March, 1982

THUR78 K.J. Thurber, "Computer Communications Techniques", Proceedings, COMPSAC78, IEEE Catalog No. 78CH1338-3C, November, 1978

THUR80 K.J. Thurber. Tutorial: A Pragmatic View of Distributed Systems, IEEE Computer Society, Catalog No. EHO-160-2, October, 1980

VAUG72 H. E. Vaughn, "An Introduction to the no. 4 ESS", International Switching Symposium Record, June 1972

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.